

Copyright

by

Joshua Murry Williams

2013

**The Dissertation Committee for Joshua Murry Williams Certifies that this is the
approved version of the following dissertation:**

**Automated Conceptual Design of Manufacturing Workcells in
Radioactive Environments**

Committee:

Sheldon Landsberger, Supervisor

Mitchell Pryor, Co-Supervisor

Matthew Campbell

Erich Schneider

Stephen Willson

**Automated Conceptual Design of Manufacturing Workcells in
Radioactive Environments**

by

Joshua Murry Williams, B.A., M.S.E.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

August 2013

Dedication

To my wife Julie and my parents.

Acknowledgements

I would like to acknowledge Sheldon Landsberger and Mitchell Pryor for their guidance, constant support, and hard work in managing the Nuclear Robotics Group. I would also like to thank Stephen Willson for his mentorship and MET-1 for their sponsorship at Los Alamos National Laboratory. This work was funded by MET-1 and Pu Sustainment at LANL.

I also want to thank the Lord for being with me throughout my academic career and the family and friends who have prayed for, encouraged, and assisted me all these years.

Automated Conceptual Design of Manufacturing Workcells in Radioactive Environments

Joshua Murry Williams, Ph.D.

The University of Texas at Austin, 2013

Supervisors: Sheldon Landsberger and Mitchell Pryor

The design of manufacturing systems in hazardous environments is complex, requiring interdisciplinary knowledge to determine which components and operators (human or robotic) are feasible. When conceptualizing designs, some options may be overlooked or unknowingly infeasible due to the design engineers' lack of knowledge in a particular field or ineffective communication of requirements between disciplines. To alleviate many of these design issues, we develop a computational design tool to automate the synthesis of conceptual manufacturing system designs and optimization of preliminary layouts. To generate workcell concepts for manufacturing processes, we create a knowledge-based system (KBS) that performs functional modeling using a common language, a generic component database, and a rule set. The KBS produces high-level task plans for specific manufacturing processes and allocates needed material handling tasks between compatible human and/or robotic labor. We develop an extended pattern search (EPS) algorithm to optimize system layouts based on worker dose and cycle time minimization using the functions and sequencing of generated task plans. The KBS and EPS algorithm were applied to the design of glovebox processing systems at Los Alamos National Laboratory (LANL). Our computational design tool successfully generates design concepts with varied task allocation and processing sub-tasks and

layouts with favorable manipulation workspaces. This work establishes a framework for automated conceptual design while providing designers with a beneficial tool for designing manufacturing systems in an interdisciplinary and highly constrained domain.

Table of Contents

List of Tables	xiii
List of Figures	xiv
Chapter 1 : Introduction	1
1.1 Nature of Nuclear Work.....	1
1.2 Design Problem.....	4
1.3 The Design Process.....	7
1.4 Scope.....	9
1.5 Objectives	10
1.6 Dissertation Organization	13
Chapter 2 : Design Background and Approach	15
2.1 Systematic Design.....	15
2.1.1 Task Clarification and Problem Formulation	16
2.1.2 Conceptual Design	17
2.1.2.1 Functional Modeling.....	17
2.1.2.2 Function-Behavior-Structure Framework	19
2.1.2.3 Generation.....	20
2.1.2.4 Evaluation	21
2.1.2.5 Optimization	22
2.1.3 Embodiment Design.....	25
2.1.4 Detailed Design.....	25
2.1.5 Design Process Scope	26
2.2 Automated Design	27
2.2.1 Representation.....	28
2.2.2 Generation.....	29
2.2.3 Evaluation	30
2.2.4 Guidance	30
2.2.5 Automated Design Scope.....	31
2.3 Computational Tools.....	32

2.3.1 Matrix Formulations	32
2.3.2 Grammars.....	33
2.3.3 Knowledge-Based Systems.....	35
2.3.4 Agent-Based Design	36
2.3.5 Catalog Design and Databases.....	37
2.3.6 Engineering Optimization.....	38
2.4 Background Summary	38
2.5 High-Level Design Approach.....	39
2.5.1 Design Needs	39
2.5.2 Computational Technique Overview	40
2.5.3 Summary	42
Chapter 3 : Concept Generation Modeling	44
3.1 System Definition	44
3.2 Function and Process	45
3.2.1 Process Planning	45
3.2.2 Robotic Task Planning.....	47
3.2.3 Process Planning Influence	48
3.3 Design Representation	49
3.4 Common Language.....	49
3.4.1 Functional Basis.....	50
3.4.2 Function Categorization.....	54
3.4.3 Terminal Function Influence.....	55
3.4.4 Language Flows	56
3.5 Function/Task Compatibility	57
3.6 Function Structure Generation.....	60
Chapter 4 : Knowledge-Based System Implementation	67
4.1 Knowledge Representation and Programming Language.....	68
4.2 Common Language Implementation.....	70
4.3 Component Database	72
4.4 Rules	78

4.4.1 General Rule Formulation.....	78
4.4.2 Embodiment Rules.....	81
4.4.3 Add Rules.....	88
4.4.4 Decomposition Rules.....	91
4.5 Knowledge-Base Additions	96
4.6 Control Program.....	97
4.7 Summary	102
Chapter 5 : Concept Generation Examples	103
5.1 KBS Input	103
5.2 Results.....	105
5.2.1 Single <i>Break</i> Example.....	105
5.2.2 Plutonium Electrorefining Example	112
5.2.3 Americium Conversion Example.....	118
5.2.4 Welding Example.....	121
5.3 Discussion	124
5.3.1 Concept Generation	125
5.3.2 Task/Process Planning.....	127
5.3.3 Limitations and Issues.....	129
5.4 Summary	131
Chapter 6 : General Layout Approach	132
6.1 Previous Work	133
6.1.1 Facility and Machine Layout	133
6.1.2 Component Layout.....	134
6.1.3 Robot and Manufacturing Workcells.....	135
6.2 General Problem Elements	137
6.2.1 Mathematical Model	137
6.2.2 Objectives	138
6.2.3 Constraints	139
6.3 Search Mechanism	141
6.3.1 Applicable Algorithms.....	141

6.3.2 Algorithm Considerations	142
6.3.2.1 Stochastic Approaches	143
6.3.2.2 Deterministic Approaches	144
6.3.2.3 Extended Pattern Search	146
6.4 Summary	146
Chapter 7 : Layout Optimization Model	148
7.1 Representation	148
7.1.1 Component Geometry	149
7.1.2 Design Variables	151
7.1.3 Configuration Information	152
7.2 Objectives	153
7.2.1 Cycle Time Minimization	153
7.2.1.1 Move Tasks	154
7.2.1.2 Other Tasks	155
7.2.2 Dose Minimization	156
7.2.2.1 Dose Derivation	157
7.2.2.2 Dose Calculation	161
7.2.2.3 Robotic Dose	164
7.3 Constraints	165
7.3.1 Within Container Bounds	166
7.3.2 No Component Collisions	167
7.3.2.1 Collision Model	168
7.3.2.2 Collision Calculation	170
7.3.3 Target Reachability	172
7.3.3.1 Path Planning	173
7.3.3.2 Obstacle Check	176
7.3.3.3 Path Generation	177
7.4 Summary	183
Chapter 8 : Extended Pattern Search Algorithm	185
8.1 General EPS Framework	185

8.2 Extensions	186
8.3 Algorithm Framework	188
8.3.1 Parameters	189
8.3.2 Preliminary Operations	190
8.3.3 Initial Configuration.....	191
8.3.4 Evaluation	192
8.3.5 New Configuration.....	192
8.3.6 Swapping.....	193
8.4 Summary	194
Chapter 9 : System Layout Examples	195
9.1 General Example.....	195
9.2 Task Allocation	202
9.3 Americium Conversion.....	204
9.4 Pu Electrorefining	208
9.5 Algorithm Performance	209
9.6 Discussion	215
9.7 Summary	217
Chapter 10 : Conclusion.....	219
10.1 Summary of Work.....	219
10.2 Future Work	222
10.2.1 KBS Knowledge Extensions.....	222
10.2.2 Concept Filtering	223
10.2.3 Generative Layout Optimization.....	224
10.2.4 Layout Model Extensions	225
10.2.5 Simulation	227
10.3 Summary	227
References.....	228
Vita.....	240

List of Tables

Table 2.1: The main factors that influence design feasibility and performance. An abbreviated list of performance metrics is presented.....	40
Table 3.1: The scale values for qualitative task criteria.....	59
Table 9.1: The <i>Moves</i> from the general example task plan.	196
Table 9.2: The parameters used for the exposure calculation.....	197
Table 9.3: The results from the layout generated in Figure 9.2.....	199
Table 9.4: The initial and final human gloveport locations.....	199
Table 9.5: Comparison of objective and penalty values for different workcells.....	204
Table 9.6: The values used to calculate the dose constant C for Am-241.....	205
Table 9.7: The task plan using manual/human labor for Am conversion.....	206
Table 9.8: The final performance of the manual Am conversion concept.....	206
Table 9.9: The performance of the automated ER workcell.....	209
Table 9.10: The results from six runs of the same task plan.....	210
Table 9.11: The computational metrics for the runs in Table 9.10.....	210
Table 9.12: Objective function values in the early iterations for Figure 9.10.....	215
Table 9.13: The gloveport progression for the layouts in Figure 9.10.....	215

List of Figures

Figure 1.1: A human [Cournoyer, 2012] and manipulators performing tasks within a glovebox.....	2
Figure 1.2: Some systems engineering domains in hazardous environments adapted from Kossiakoff [2011].....	6
Figure 1.3: The steps of the design process with emphasis placed on the conceptual and embodiment stages. Stage boundaries for two other formulations relative to the first are shown.....	8
Figure 2.1: An example function structure for an electromechanical device [Helms, 2009].	20
Figure 2.2: The portion of the design process this work addresses and the automated aspects.	27
Figure 2.3: Flowcharts for computational synthesis stages from Campbell [2003a] (left) and Shea [2003] (right).	28
Figure 2.4: The general components of a knowledge-based system.	35
Figure 3.1: The function and flow classes of the Functional Basis [Hirtz, 2001].	52
Figure 3.2: The eight function classes and their non-terminal and terminal functions.....	53
Figure 3.3: Common manufacturing tasks for humans and robots [Ghosh, 1986].....	54
Figure 3.5: An example of function structure generation.	63
Figure 3.6: An alternate function structure generated if <i>Press</i> embodies <i>Break</i>	64
Figure 3.7: The final function structures and task plans of Figure 3.5 and Figure 3.6.	65
Figure 4.1: Flow and implementation of design knowledge.	68
Figure 4.2: The FUNCTION class template, the CHANNEL and MOVE sub-classes, and instances of the MOVE class called <i>Move-1</i> and <i>Move-2</i>	71

Figure 4.3: A muffle furnace [www.coleparmer.com], an induction furnace, and a hot plate [www.enasco.com].....	73
Figure 4.4: The high-level FLOW class and some sub-classes.	75
Figure 4.5: Two OPERATOR sub-classes and some flow instances.	76
Figure 4.6: Some classes for processed flows and particular instances.....	77
Figure 4.7: The general sequencing of rule application.....	80
Figure 4.8: An example operator embodiment rule.	82
Figure 4.9: An example of the LHS of rules interacting with instances in the CLIPS environment.	84
Figure 4.10: The changes caused by rule firing in CLIPS code (top) and the function block representation (bottom).	86
Figure 4.11: The <i>Break</i> embodiment rule.	87
Figure 4.12: The initial and final states after the <i>EMBODY-break</i> rule is applied.	88
Figure 4.13: An add rule that inserts an <i>Open</i> function.	89
Figure 4.14: The results of firing the <i>add-OPEN-con</i> rule for a <i>Heat</i> function.....	90
Figure 4.15: A decomposition rule for <i>Transport</i>	92
Figure 4.16: The results of firing the <i>dec-TRANSPORT</i> rule.	93
Figure 4.17: A decomposition rule for <i>Unload</i>	95
Figure 4.18: Three different decompositions for <i>Unload</i> depending on the involved flows.....	96
Figure 4.19: An example of a MDESIGN object.	98
Figure 4.20: The CLIPS and C++ implementations for our KBS.	99
Figure 4.21: An example of passing functions between C++ and CLIPS.	101
Figure 5.1: The string input for the <i>Break</i> function and a block representation.	105
Figure 5.2: Some concepts for breaking a composite utilizing only human labor.	106

Figure 5.3: Some automated concepts for breaking materials apart.....	110
Figure 5.4: An opened furnace cell (left) and the resulting solid from an ER run (right). The Pu is the outermost ring, followed by the crucible (white) and inner anode.	113
Figure 5.5: The input for the Pu ER example.	114
Figure 5.6: A manual concept generated for the input in Figure 5.5.	115
Figure 5.7: A concept utilizing robotics for the input in Figure 5.5.	116
Figure 5.8: Oxide conversion operations pictured for Pu [LANL, 2008b].	118
Figure 5.9: The input for americium oxide conversion.	119
Figure 5.10: A manual design for americium conversion and mixing.	119
Figure 5.11: An example layout for <i>Concept 1</i> in Figure 5.10.	120
Figure 5.12: The robot handling a convenience can (left), holding a DOE standard inner can (center), and positioned next to the automatic welder (right) [LANL, 2008a].	122
Figure 5.13: Input for placing a convenience can inside another can then welding the outer can.	122
Figure 5.14: A generated task plan for a robot performing packaging in the ARIES line.	123
Figure 5.15: A concept for a robotic automated welding workcell.	124
Figure 6.1: Computational approaches to layout problems on a continuum from deterministic to random [Cagan, 2002].	142
Figure 6.2: Some example patterns and their mapping onto a lattice [Torczon, 1997a].	145
Figure 7.1: Representations of component geometry for box and cylindrical shapes with example component instances below.	149

Figure 7.2: A 2-D protection of 3-D component geometry on the glovebox floor (x-y plane).....	150
Figure 7.3: A box component rotated by θ and two container locations (1 & 2) with CLIPS instance representations.	151
Figure 7.4: Calculating robot and human dose for two different <i>Moves</i> in the task plan.....	163
Figure 7.5: Calculating the penalty for component boundary violations.....	167
Figure 7.6: The simple component collision models from a 2-D perspective (1 – 4) and an instance and 3-D representation for a component similar to (3).	169
Figure 7.7: A collision between two components (top) and the same 2-D projection circle intersection but with no 3-D collision (bottom).....	171
Figure 7.8: The locations of the human (left) and robot (right) centers and wrist points.....	173
Figure 7.9: An EEF trajectory as a series of via points containing orientation information.....	174
Figure 7.10: A serial manipulator reaching around an obstacle to locate its wrist point at a component's access point.....	175
Figure 7.11: Detecting an obstacle between the operator and access point in 3-D (left) and 2-D (right).	177
Figure 7.12: The path formulations for going around (left) and above (right) an obstacle with a single circle.	178
Figure 7.13: Two views of example 3-D paths for going over and around an obstacle that has a single circle.	179

Figure 7.14: Extending an obstacle's corners in the 2-D plane (top) and determining the intermediate points for paths over (left) and around the obstacle (right).	180
Figure 7.15: Two views of example 3-D paths for going over and around an obstacle that has multiple circles.	181
Figure 9.1: The initial configuration for the layout in Figure 9.2.....	196
Figure 9.2: Two different views of the layouts for the task plan in Table 9.1.....	198
Figure 9.3: Another layout generated for the task plan in Table 9.1.	200
Figure 9.4: The initial and final locations for a manual workcell.....	203
Figure 9.5: The initial and final locations for a robotic workcell.	203
Figure 9.6: The final layout using manual processing for Am conversion.....	205
Figure 9.7: The layout for an automated Pu ER workcell.	208
Figure 9.9: The objective and penalty function value trends for Am conversion.....	213
Figure 9.10: The progression of selected component locations and orientations after the first two iterations for <i>Run 5</i>	214
Figure 10.1: The abbreviated input and output of our conceptual design tool.	221

Chapter 1 : Introduction

System design covers a wide variety of subject areas and expertise. The unique design problems that result vary in complexity but utilize similar proven techniques to arrive at effective solutions. As more interdisciplinary work and research is performed, the ability of design engineers to obtain and utilize knowledge from multiple domains becomes increasingly important. The success of a design hinges on the design engineers' ability to formulate the requirements, constraints, and existing solution frameworks from all associated disciplines to fabricate a solution that satisfies all domain experts who have input on the design.

Nuclear design applications in particular are highly interdisciplinary and present complex design problems. Due to the nature of nuclear design, it may benefit from computational techniques that automate or automatically execute various aspects of the design process. Applying such techniques to nuclear design leads to interesting propositions for automated design, particularly at the conceptual level.

1.1 NATURE OF NUCLEAR WORK

There are a number of nuclear processes that must be performed in hazardous environments. These include nuclear fuel fabrication, weapons manufacturing, material separation and conversion, waste cleanup, decommissioning, and material surveillance and monitoring. In particular, manufacturing tasks (ex: casting, joining, machining, etc) typically utilize processing methods and machinery identical to comparable processes involving non-nuclear materials [LANL, 2012]. The desired sub-tasks are also very similar to typical manufacturing processes and include material movement and positioning and apparatus and tool operation. Functional and product requirements are

similarly stringent with specifications on material quality, composition, dimensions, and tolerances. In the nuclear domain, however, the nature of the materials processed and the working environment lead to additional constraints, considerations, and obstacles that complicate the ability to manufacture.

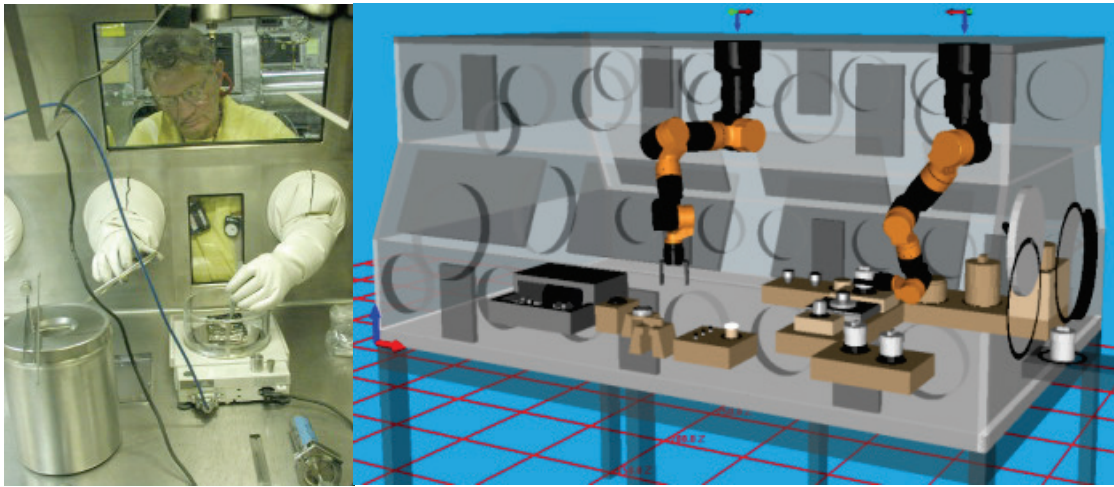


Figure 1.1: A human [Cournoyer, 2012] and manipulators performing tasks within a glovebox.

The presence of radiation and the potential to spread contamination dictate many work parameters. Most processes are conducted within sealed, stainless-steel gloveboxes to provide a level of containment and radiation shielding for workers (Figure 1.1). The dose received by human workers is of particular concern. Dose limits are set to restrict workers to dosages safely below the high levels known to cause deterministic effects. At lower doses, however, the exact effect of dose on worker health is hard to quantify. The current accepted model assumes that with each small dose, the probability of observing a detrimental effect linearly increases. Because of this, exposure to radiation should be

minimized during processing. This is the idea behind the ALARA principle – that radiation doses should be kept “As Low As Reasonably Achievable” considering economic, social, and other factors.

Traditional techniques for reducing worker dose include shielding, increased separation distance from a radioactive source, decreased time in the environment, and improvements to processing techniques. However, some materials and processes are too radioactive/hot for radiation workers even when the mentioned protection measures are implemented. These cases require automation, robotics, and/or other remote handling tools to adequately lower worker hazard.

The environment may also contain non-radiological hazards that can potentially injure workers and lead to operational work stoppages. These include mechanical hazards such as moving parts and sharp objects, electrical and chemical hazards, and unfavorable ergonomic conditions, particularly those stemming from glovebox work. Due to these hazards and compatibility issues with environmental conditions, in some cases, ideal processing techniques and components cannot be utilized, leading to less favorable manual tasks and hindered productivity.

Nuclear work is also susceptible to administrative barriers of production. The work must adhere to a budget and remain financially feasible. While ALARA considerations are required, there is a point where the financial cost of implementing additional protection means does not justify the marginal decrease in dose provided (assuming doses are well below their limits). Along this line, the International Commission on Radiological Protection (ICRP) provides suggested techniques for radiation protection that include justification of work, optimization, and application of dose limits [ICRP, 2007]. Nuclear Regulatory Commission (NRC) regulations, in particular 10CFR20 “Standards for Protection Against Radiation”, must also be followed

in addition to site-specific rules that may impose further constraints. These can all lead to limits on the amount of materials that can be processed, the waste that is generated, and the amount of labor required.

The NRC is currently weighing options to adopt in 10CFR20 the new occupational dose guidelines in ICRP Publication 103 [Cool, 2012]. This includes a recommended occupational effective dose of 20 mSv averaged over five years with no more than 50 mSv in any one year [ICRP, 2007]. The current dose limit in 10CFR20 is a fixed dose limit of 50mSv per year. The majority of stakeholders have expressed opposition to such changes based on the difficulty to achieve the more restrictive dose limit [Cool, 2012].

1.2 DESIGN PROBLEM

The restrictive nature of nuclear work highly constrains manufacturing system design. Additionally, various objectives must be addressed and analysis is complicated by the interdependence and tradeoffs of design decisions. In addition to typical manufacturing objectives such as high efficiency, low cost, high throughput, and safety, the unique working environment introduces uncommon and domain specific objectives such as dose minimization, favorable ergonomics, and hazardous waste minimization. These objectives are directly influenced by how operations are performed which heavily relies on the selection of material handlers: human workers, robotics, or other automation. The feasibility of automation and/or robotics for a particular sub-task requires an understanding of what automated capabilities exist and how to determine task compatibility. Based on the hazard level, operator/task compatibility, and the trade-offs

between human and robotic labor, a system design may utilize both types of material handlers.

In a shared workspace, many factors determine the potential ease and benefit of utilizing automation. From a task standpoint, the usefulness of robotics depends on the apparatuses and other components it can use, its interaction with the human operator, and the configuration of the components within the glovebox. There are also constraints on which material and component types are compatible with each other and the radioactive environment which can further limit implementation effectiveness. Designers must also address automation reliability, failure scenarios, and the human safety concerns and non-radiological hazards that arise from sharing the workspace with a machine.

Thus, manufacturing system design in the nuclear domain is complex and requires multidisciplinary knowledge with considerations relevant to mechanical, electrical, and nuclear engineering, chemistry, materials science, and safety. In essence, this is a systems engineering problem where some level of knowledge from many fields is necessary for effective design (Figure 1.2). The knowledge depth requirement for the systems engineer in each domain may vary. Conceptualization of more creative and perhaps better designs can be hindered by a lack of knowledge in any field. For instance, remote handling techniques may be less familiar to some designers, and if newer technologies are not well-known, less obvious and more promising options can be overlooked. Furthermore, if communication of requirements between disciplines is inadequate, this can lead to setbacks as incompatibilities or infeasible ideas are not realized until further downstream in the design process.

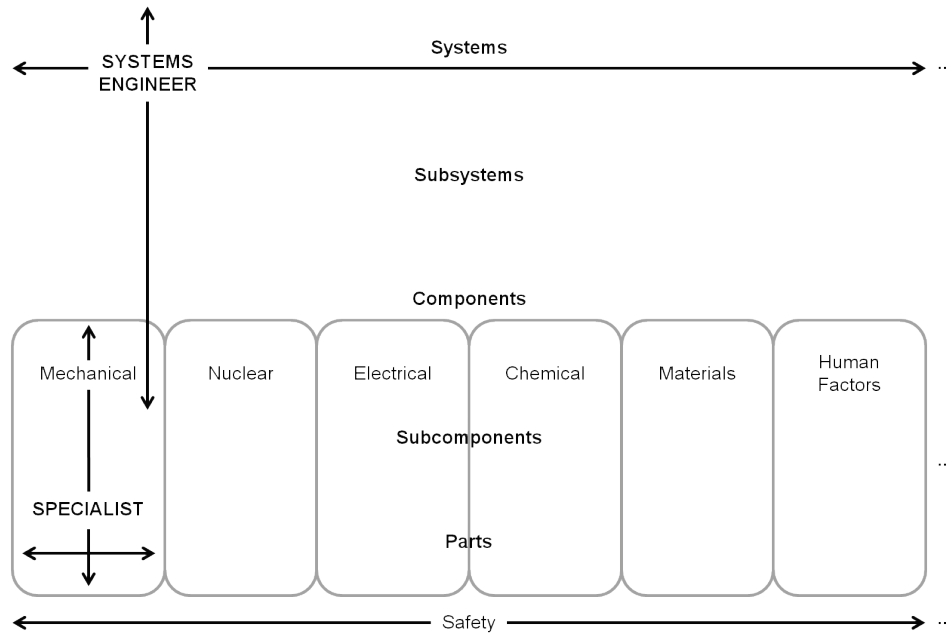


Figure 1.2: Some systems engineering domains in hazardous environments adapted from Kossiakoff [2011].

There is an urgent need to develop systems that can meet the demands of nuclear manufacturing. As previously mentioned, there is anticipation of stricter occupational dose limits in the near future. There can also be agency or site initiatives to lower radiation workers' dosages even if dose limits remain the same. Additionally, many systems in the nuclear processing infrastructure are old and outdated and in need of updates or replacement. Some operations continue to ramp up in scale and existing systems that utilize manual processing are not capable of meeting future demand in addition to their inability to provide adequate radiation protection under new regulations.

1.3 THE DESIGN PROCESS

This design problem may be complex, but can be handled by the existing framework of mechanical engineering design. Typically, designers follow a systematic approach to steer toward an effective and quality design. Different structured approaches have been proposed for engineering design [Roth, 1981][Hubka, 1982][Suh, 1990][Ullman, 1992][Otto, 2001][Pahl, 2007]. Although researchers may define different bounds for each design stage or have different numbers of steps, the underlying tasks and order are generally consistent [Sim, 2003]. One formulation that has found extensive application in research is the systematic approach of Pahl and Beitz [2007][Hundal, 1990][Dym, 1994][Dieter, 2000]. This formulation is typical of the German approach to design and consists of four stages: task clarification, and conceptual, embodiment, and detailed design (Figure 1.3). Overall, the design process performs a transformation from an abstract plan with process requirements and constraints to a physical system capable of executing the process to specifications.

The task clarification stage involves defining many of the general requirements and constraints of nuclear work laid out in the preceding sections. These govern conceptual development as the design progresses. Conceptual design utilizes creative and inventive techniques to generate various design alternatives/concepts. This stage includes process/functional modeling that uses various levels of abstraction to decompose the high-level design concept into smaller and easier to work with units. This typically means defining the overall system function and decomposing it into smaller sub-functions. The embodiment phase seeks to associate components to functions and add form and more concrete details to the physical structures. This involves defining system topology or connectivity and preliminary component modeling/features, material selections, and spatial configurations. During the conceptual and embodiment stages, evaluation checks

are made to confirm feasibility and determine performance. Such checks and subsequent refinements to requirements can suggest design changes that produce iterative generation, evaluation, and refinement loops in the early design stages. Conceptual and embodiment design or only the latter are sometimes referred to as preliminary design [Dieter, 2000][Arora, 2012]. Once a definitive design is selected, detailed design adds exact component geometries, tolerances, spatial configuration, etc, to allow for actual system construction and operation. In the case of manufacturing, this includes detailed part fabrication instructions and processes. Additional iterative refinements may also occur here. This design process is addressed in more detail in Chapter 2.

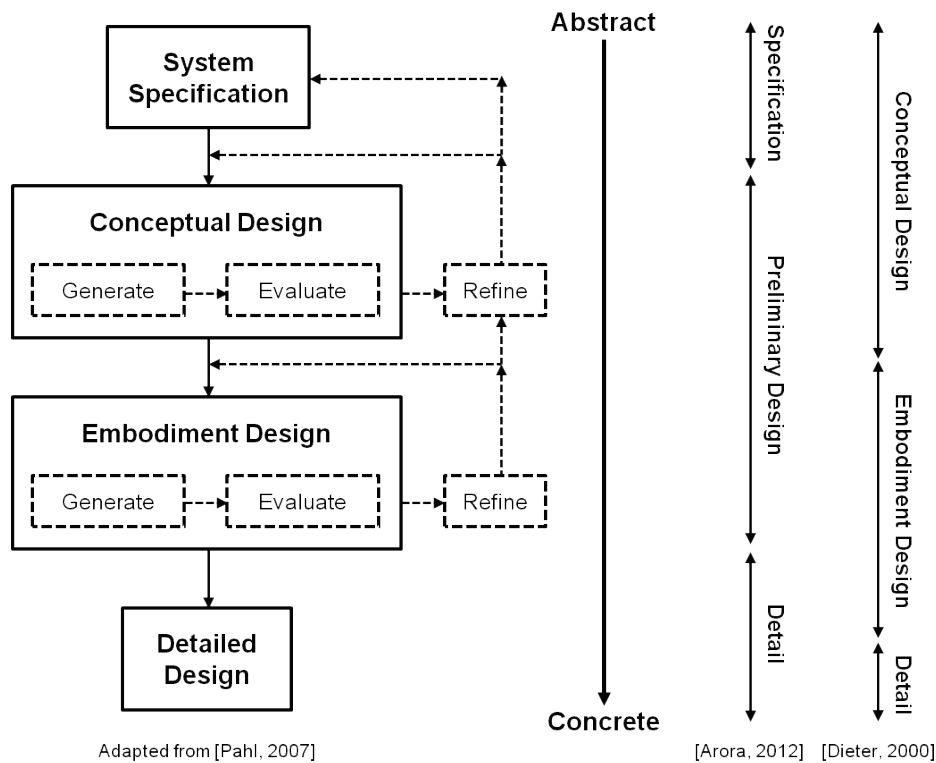


Figure 1.3: The steps of the design process with emphasis placed on the conceptual and embodiment stages. Stage boundaries for two other formulations relative to the first are shown.

1.4 SCOPE

The design process described above provides a baseline for design work in the nuclear domain. The scope of our work is defined relative to the engineering design process and the application domain. In this context, we identify a number of bounds that condense our work into a more manageable form:

- The emphasis is on system design (Figure 1.2). These systems consist of components and operators working together to execute a process.
- Systems are designed around pre-defined processes with a set number of required sub-tasks. Any improvement to processing results from operator and component selection and not from direct analysis of the process itself.
- This work is much more conceptual in nature. In regard to systematic design, we focus on aspects of the stages before detailed design.
- We adapt the existing design methodology and techniques for nuclear design and do not develop an entirely new process design algorithm.
- We consider material handling and manufacturing inside the confined and controlled environments of gloveboxes.
- Emphasis is placed on feasible and optimized system configurations and not on optimization of individual robotic or mechanical components.

Although we narrow our focus to nuclear workcells where humans and robots work together, automation/robotics could be incorporated into system design and operations any time reduction or elimination of occupational dose is desired and/or manufacturing goals such as reduced cycle time and increased product quality can be realized from automated operations. Our techniques are generally applicable to many types of manufacturing processes and their systems, especially since many non-nuclear work environments are less constrained. Essentially, the problem addressed here can be

generalized to any application area where robotic systems should be included in possible solutions to engineering problems. The nuclear domain provides fertile ground to address this problem:

- Applications requirements in the nuclear domain are further afield than those of many other domains. Thus, system designers may be familiar with the breadth of multidisciplinary requirements but are unlikely to have the expertise to make important design decisions in all necessary areas.
- Dosage in particular provides a complex yet quantifiable design metric that will allow this effort to focus on methodology and objectively measure its relative effectiveness.
- While complex, the confined nature and limited number of objects in the environment provide the opportunity to test proposed methodologies with realistic systems.

1.5 OBJECTIVES

The conceptual design stage is particularly important. It affects future decisions and poor initial choices can lead to few and possibly inferior options later on. Our goal in conceptual design is to explore all feasible design options. We do not want to limit ourselves to certain design structures/shells. We focus on creative selections for operators (i.e. material handlers) due to the hazardous environment. Often, in existing systems, we see either manual processing or some sort of automated solution, and the robotics utilized is very basic or old technology. We not only want to generate solutions that utilize more advanced technology, but also consider those that mix both manual and automated processing.

From a process perspective, the selected material handlers/operators and components affects system feasibility and sub-functions. Certain operators may more easily manipulate specific components or restrict the feasible component set. To identify task compatibility for a selected operator, we need some physical knowledge of components and depending on the selected component additional sub-functions may be necessary for the operator to use that component. Important constraints also stem from how the operator can move within the workspace layout given component geometries and locations. An operator may be compatible with task requirements, but this is nullified if the operator cannot reach the task point and/or collides with other components on the way to the task or while performing the task. Thus, some semblance of structure is needed to address feasibility and identify all system functions. Our overall goal is to develop designs with feasible operator and component combinations using approximated requirements, components, trajectories, and layouts. The result is a mixing between conceptual and embodiment design.

Thus, conceptualizing designs without any knowledge of probable structure is not as effective as utilizing this information to guide concept generation. However, this information could be difficult to obtain. In our case, the design domain may be new, but we have a design baseline of systems designed in non-radiological environments performing similar processes. Furthermore, in this domain, if we want to incorporate more automation and robotics, we have the old manual processing as a baseline to derive requirements and reference system components and configuration. Although designs may vary within a single domain or between domains, desired functions can typically only be achieved in a limited number of ways [Qian, 1996]. Thus, extraction of generalized design knowledge from analogous designs can help lead to new creative designs.

We can also incorporate design knowledge such as previous solution structures, typical processing/manufacturing functions and sub-tasks, broad operator (human/robot) capabilities, and general component types and approximated structure into conceptual design. We can also derive qualitative and quantitative requirements from this information. In this way, we utilize prior *component* and *processing* knowledge for creative *material handling* selection in conceptual designs. Without this information, and considering the high number of constraints, conceptual design may be ineffective as decisions advance toward a design that is later found to be infeasible.

This framework is only possible and effective if knowledge from multiple domains is known and has a means of exchange and utilization for decision making. This proposed execution of conceptual and embodiment design can be aided by computational design tools. This particular problem lends itself to Automated Design Synthesis (ADS). ADS draws on the computational abilities of computers in order to ease the burden on the human designer [Campbell, 2003b]. At present, automated design has not been applied to manufacturing system design to the extent which it has for product design. ADS can alleviate the problems encountered in this design task such as storing knowledge and dependencies from multiple disciplines and handling tradeoffs from competing objectives.

As such, there is also no systematic and automatic technique guaranteed to generate feasible results. We will work toward this end. In this work, ***we incorporate traditional engineering design principles into a computer-based tool to automate the synthesis of conceptual manufacturing system designs in the nuclear domain.*** In our scope, this work has a number of goals:

- Extraction and storage of general information about manufacturing, material handlers, and component structure to utilize in conceptual design

- Development of qualitative and quantitative criteria for characterization and evaluation of operator (human and robotic) task compatibility early in the design process
- Framework to store and recall expert knowledge from multiple domains for more effective communication of design requirements and better exploration of the design space
- Development of a structured, generalized, and flexible computational design tool that utilizes domain knowledge to execute conceptual design similar to human experts
- Validation of our framework through application to design scenarios in the nuclear domain

This work will construct a formal technique for conceptual design in the nuclear domain utilizing a computational design tool. The framework will be easy to extend and interface with detailed design techniques. The result will be the generation of better concepts for detailed design and a smooth transition from conceptual to concrete. Overall, more structured, creative, and efficient nuclear system design is possible.

1.6 DISSERTATION ORGANIZATION

This thesis will start with an overview of the systematic approach to engineering design and a description of automated design synthesis and its associated computational tools (Chapter 2). Based on the described design problem and existing design techniques, we will propose a system model (Chapter 3) that is utilized by a knowledge-based system (KBS) to generate design concepts (Chapter 4). Then we will present examples of applying our concept generation technique to glovebox processes at LANL (Chapter 5).

We then discuss the layout of design concepts for feasibility and evaluation (Chapter 6). The layout problem is formulated as an unconstrained optimization problem (Chapter 7) that performs searches using an extended pattern search (EPS) algorithm (Chapter 8). This algorithm is applied to designs generated by our KBS (Chapter 9). We finish with a conclusion to our work and potential extensions to our KBS and EPS algorithm (Chapter 10).

Chapter 2: Design Background and Approach

This work is interdisciplinary, requiring knowledge from several different backgrounds. Taken within the context of mechanical engineering design, our problem is more manageable. We will begin with a description of the mechanical design process and techniques and then describe how this process can be managed by a design framework and computational tools. For our work, the focus is to build on accepted methods that have been implemented into computational design tools and not derive a new design methodology. An understanding of these methods and tools is essential for executing the objectives of this work. Furthermore, this dissertation is intended to be useful for both a mechanical and nuclear engineering audience. Thus, some of the more common design paradigms applied as a part of this research will be presented in sufficient depth for a technical but not necessarily expert reader.

2.1 SYSTEMATIC DESIGN

There are two categories of design methodologies: descriptive and prescriptive [Finger, 1989]. Descriptive methods study what processes and strategies designers utilize to design and also build models of the designer's cognitive process. Prescriptive methods seek to prescribe how the design process should proceed or what attributes a design artifact should have. Both areas are relevant to the implementation of a successful artificial intelligence technique for automating design.

For our work, we utilize the design methodology of Pahl and Beitz [2007]. This prescriptive formulation consists of the four stages presented in Chapter 1: task clarification, and conceptual, embodiment, and detailed design. This is an iterative

process of generation, evaluation, and refinement starting with an abstract concept and leading to a concrete, physical system.

2.1.1 Task Clarification and Problem Formulation

The process begins with the identification of a design need for a product or system. The essential design task or problem is formulated and clarified through the development of a requirements list. This accounts for customer/stakeholder needs, and functional, operational, product, process, and other requirements. A common tool for deriving product requirements is Quality Function Deployment (QFD) [Hundal, 1990][Dieter, 2000] which is also known as *House of Quality* [Hauser, 1988]. This chart-type tool utilizes relationships, importance information, and other assessments to help transform customer requirements into target engineering characteristics for a product.

A comprehensive requirements list establishes the bounds for guiding the design process and helps develop the product characteristics needed to achieve those requirements. In order to develop a feasible design, the system attributes must be selected in such a manner that all requirements are attained while no constraints are violated. A feasible design can then be rated according to its performance for a number of desirable metrics. Requirements reformulation and design modifications can be guided by estimates and tradeoffs of performance during various parts of the design process. A poor requirements list can lead the design in unfavorable directions, wasting effort, money, and possibly ending with a design that fails to meet customer expectations.

2.1.2 Conceptual Design

Creative conceptual design requires an initial concept description that is not tied to any particular solution types or structure. Designers employ various layers of abstraction to steer away from a strict and specific concept description to find the crux of the design problem [Pahl, 2007]. Functional design is often utilized, as is the case for the structured approaches previously mentioned. Functional modeling can link higher and lower levels of system design and description and provide a common method of communication between multiple disciplines [Erden, 2008]. Working from this level of abstraction, the design concept is initially described by the overall function it performs. This is in contrast to a physical design approach where the initial concept is directly decomposed into subassemblies and components [Dieter, 2000]. In this case, the design is assumed to take on a particular physical form and the design task becomes selecting among one or more categories of solution types or from a component catalog. Functional modeling of concepts helps remove bias and promote generality in the early stages of design.

2.1.2.1 Functional Modeling

Product or system function is defined in many ways [Chandrasekaran, 2005]. Function can be seen as a subjective description that links intentions in the subjective realm to behaviors and structures in the objective realm [Erden, 2008]. Two representations include natural language descriptions and mathematical representations [Chakrabarti, 2001]. Natural language descriptions use verbs to describe what a structure does or “the teleology of the object, i.e. what it is for” [Gero, 2004]. Mathematical representations define function as a transformation between input and output or more

specifically “the intended input/output relationship of a system whose purpose is to perform a task” [Pahl, 2007].

Functional modeling for an artifact or manufacturing process can take a process-centered or device-centered view [Umeda, 1995][Kitmura, 2004]. The process-centered view focuses on a physical process that changes components’ attributes while they participate in the process [Forbus, 1984]. The device-centered view models a design as a network of components with input and output flows of materials, energy, and/or signals/information. Each component is typically viewed as a “black box” that performs its function to transform input flows into the desired output flows. In this way, a device or system is seen as “black box modules connected with input-output relations” where system agents “process their own input data and produce outputs to be transferred to the other agents” [Erden, 2008].

Commonly, descriptions of function are given as verb-noun pairs which come from value engineering [Nagel, 2009]. The function is a verb acting on a noun which is a flow. Efforts have been made to standardized the terms (verbs and nouns) utilized for function descriptions [Altshuller, 1988][Hundal, 1990][Ullman, 1992][Hirtz, 2001][Pahl, 2007]. The goal of these formulations is to define a list of terms that can describe most physical functions from analysis of products, processes, experience, etc. Some functional terms stem from “generally valid” functions [Pahl, 2007] that are categorized by their common input and output combinations. Flows are generally considered to be that of materials, energy, or information/signal. Functional ontologies take such classification a step further, specifying the terms that conceptualize a domain and the rules for combining and relationships between those terms [Kitamura, 2004][Lemaignan, 2006].

2.1.2.2 Function-Behavior-Structure Framework

Rinderle [1991] states that “design can be viewed as a transformation from a specification of design requirements in a function language to a description of an artifact expressed in a physical language.” Starting from a functional description, design representation commonly follows a function-behavior-structure (FBS) formulation [Gero, 1990][Gero, 2004][Helms, 2009]. Using this representation, the overall, high-level system function is defined then decomposed into lower-level sub-functions. These sub-functions are less abstract than the function they decompose. Function decomposition occurs until physical structures (i.e. components) with behaviors that can realize that function can be identified. The combination of all the sub-functions is referred to as the *function structure* representing the overall function [Pahl, 2007]. The behaviors are physical phenomena and broadly represent a physical, chemical, or biological process. In the case of physical processes, the behaviors are related to physical laws that govern the physical properties involved [Pahl, 2007]. The behaviors “describe the working principles that realize the functions from a physical and component independent point of view” [Helms, 2009]. Once behaviors are selected to satisfy all the sub-functions, concrete components must be selected that embody the given behaviors. This generates a design concept architecture (see Figure 2.1) that can suitably accomplish the desired behaviors and thus fulfills the required functionality [Helms, 2009]. Using the FBS approach, a transition is made from a functional description of an artifact or system to a physical architecture capable of performing the desired functions based on the defined requirements.

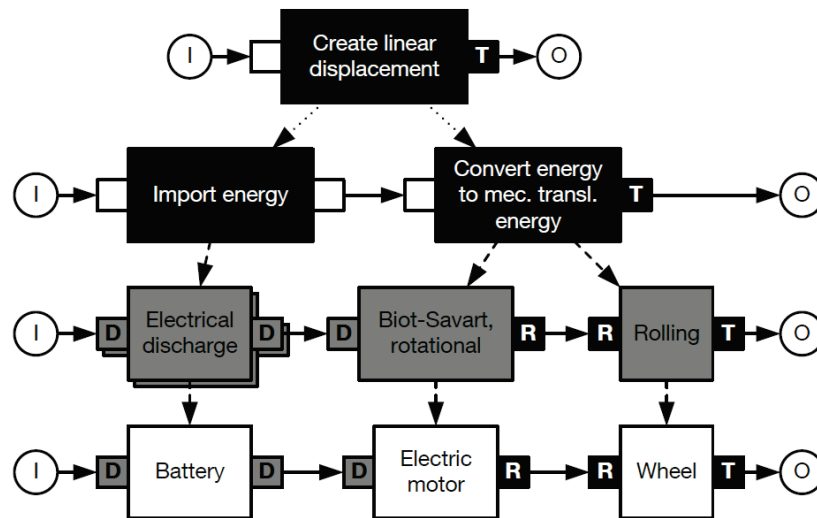


Figure 2.1: An example function structure for an electromechanical device [Helms, 2009].

2.1.2.3 Generation

There are different methods for generating design concepts, some intuitive and some more directed/structured. Traditional intuitive techniques that involve interactions among a design team include brainstorming, mind mapping, synectics, and the 6-3-5 process [Dieter, 2000][Otto, 2001][Pahl, 2007]. Other techniques rely on known information to direct the generation of new ideas. The Theory of Inventive Problem Solving (TIPS) [Altshuller, 1988] outlines forty different inventive principles for resolving system conflicts and improving concepts. Design by analogy utilizes analysis of an analogous problem or system with similar function and/or behaviors to search for solutions [Qian, 1996][Pahl, 2007]. Design knowledge can also be obtained from patent searches, trade journals, handbooks, and catalogs [Otto, 2001][Ullman, 1992][Dieter, 2000]. Knowledge engineering is the process of extracting knowledge from experts through extensive interviews over a long period of time [Giarratano, 2005]. Reverse

engineering, in which products are systematically disassembled and analyzed, is also a means to gain functional and physical information for building a knowledge-base [Otto, 2001].

In the context of functional modeling, eventually concepts that describe *how* a function will be performed must be identified. These are called “working principles” by Pahl [2007] and “reflect the physical effect needed for the fulfillment of a given function and also its geometric and material characteristics.” A design concept or variant is found by combining the solutions to each sub-function. Different designs result when there are multiple means of realizing a given sub-function.

Concept variants are commonly generated using a morphological matrix [Zwicky, 1948]. The matrix rows contain the sub-functions of the function structure. Each column contains behaviors or working principles that can satisfy the row’s sub-function. These are discovered utilizing the concept generation techniques described above. A feasible design concept/variant consists of one working principle from each row where all principles are compatible. Morphological matrices can “identify novel combinations of existing solution principles” and “reveal unorthodox combinations” [Olvander, 2009]. Through this process, form begins to be associated to function.

2.1.2.4 Evaluation

Generated concepts must be evaluated to determine if they deserve further development. To make comparisons and decisions, the concepts must be described in the same language and exist at the same level of abstraction [Ullman, 1992]. Depending on the maturity of concepts, data and information may exist at different complexity levels. Concept information may be derived from a qualitative or non-numeric scale or may be

measurable with associated units. Evaluations may be absolute where concepts are evaluated for feasibility against established requirements or relative where concepts are compared with each other [Ullman, 1992].

Evaluation methods also exist at different complexity levels to handle concepts with varying abstractions of information [Otto, 1995]. Pugh selection charts allow non-numerical comparisons of concepts with a datum concept using criteria that are typically obtained from QFD and the requirements list [Ullman, 1992][Otto, 2001]. A dominance matrix can be utilized to make pair-wise comparisons of each alternative in terms of one objective when minimal information is known about concept attributes [Otto, 2001][Triantaphyllou, 2000].

Often, multiple criteria or objectives must be computed for concepts. Weighted decision matrices determine an overall concept score using weighting factors and criteria values derived from an established numerical point scale. The Analytical Hierarchy Process (AHP) “uses a series of pair-wise comparisons...to determine the relative performance of each alternative in terms of each of the decision criteria” [Triantaphyllou, 2000]. In multi-criteria cases, each criterion must be converted to the same type of value in order to combine performance information into a single score, allowing both subjective and numerical factors to be utilized together [Dieter, 2000]. Utility theory also provides a technique for converting criteria values to a common scale and better incorporating customer preference into criteria scores [Thurston, 1991].

2.1.2.5 Optimization

The described generation and evaluation techniques are suitable during conceptual design when detailed and/or numerical data is lacking. Optimization

techniques are more applicable to concepts with more concrete parameters and contain additional functionality for design generation and evaluation. In our work, gradient based optimization techniques are not as applicable since we are not likely to have an analytical function and existing derivatives for one or more objective functions. We will therefore focus on non-gradient based or direct search methods. There are several *optimization engines* [Arora, 2012] or frameworks that can perform direct search optimization problems:

- Pure random search – This stochastic method relies on random numbers to explore the search space. An extension of this technique is the *multistart method* [Arora, 1995].
- Simulated annealing (SA) – This technique randomly generates new solutions in the neighborhood of the current design. New solutions are accepted if they improve the objective function, while inferior solutions may be accepted with a probability given by the Boltzman-Gibbs probability density function [Kirkpatrick, 1983].
- Tabu search – This search builds a historical record of investigated feasible solutions and uses it to investigate new solutions and escape local minima [Arora, 1995].
- Genetic algorithms – These utilize the mechanics of natural selection to produce new and better populations that progress toward the optimal. Each generation, the most fit are mated and variation is introduced by crossover and mutation operators [Arora, 1995].
- Pattern search – This method uses a set pattern to make exploratory moves about a current solution, accepting a tried solution if its objective function is improved over the current solution [Torczon, 1997a].

Hybrid methods, which incorporate at least two methods, also exist. These include techniques where two methods are used sequentially, work on different subsets of the solution space, or are applied at different levels of the optimization [Andersson, 2000].

In our design problem, we typically have to work with multiple objectives. This leads to the idea of searching for non-dominated or Pareto optimal solutions. “A point is Pareto optimal if there is no other point that improves at least one objective function without detriment to another function” [Marler, 2004]. From the early stages of design, we usually have customer preference information to determine the relative importance of each objective before optimization. This allows us to derive a single score based on an aggregation of all objectives similar to the multi-criteria evaluation techniques previously discussed. This is equivalent to focusing on a specific solution on the Pareto front. Some aggregation approaches include:

- Weighted sum approach – Different weights are given to the objectives and aggregated to form a single figure-of-merit. Often, normalization or other transformations are needed for each objective function to allow for aggregation/scalarization. A similar technique is the weighted global criterion method which combines all objectives into a type of utility function [Marler, 2004].
- Goal programming – Desired goal criteria are formulated for each objective function, and the total deviation from this utopian set is minimized.
- Fuzzy logic approach – Uses the concept of fuzzy sets and logic to obtain transformed values of the objective functions which through logic yields an overall value of the selected set of parameters [Andersson, 2000].

Optimization principles and algorithms can be utilized throughout the entire design process depending on the state of information and design task.

2.1.3 Embodiment Design

Embodiment design adds form to a concept and develops its architecture or the arrangement of its physical components. This involves preliminary selection of component dimensions, materials, and layout. The latter includes determining component connectivity and spatial positions relative to one another and their environment. This phase makes use of estimates and rules of thumb that reflect the designer's experience [Dym, 1994]. The preliminary layout is evaluated and refined to develop a definitive layout [Pahl, 2007] through iterative cycles of generation and evaluation similarly performed in the conceptual stage (see Figure 1.3).

In FBS modeling, the embodiment stage involves the transition from behaviors that fulfill functions to preliminary structures/components that embody those behaviors. Often, it is difficult to make a distinction between the physical effect and the form features [Pahl, 2007]. Therefore, when generating concept variants, a sub-function solution typically has some semblance of form (ex: rough geometry or material characteristics) linked to the physical process. Thus, the line between conceptual and embodiment design is often blurred and a similar type of transition also occurs between embodiment and detailed design.

2.1.4 Detailed Design

As generation and evaluation iterations lead to design refinements in the embodiment stage, a definitive design(s) emerges that will be carried on to completion. Detailed design consists of two main tasks: determining the final parameter values and specifying the best manufacturing tolerances [Ullman, 1992]. This results in detailed (CAD) drawings with dimensions and tolerances, parts lists, and documentation of fabrication/manufacturing information. Much of the relevant knowledge for detailing is

expressed in very specific rules as well as in formulas, handbooks, algorithms, databases, and catalogs [Dym, 1994].

Determination of final parameters is aided by analytical models and optimization. Mathematical models are abstractions that describe an artifact using mathematical expressions of relevant natural laws, experience, and geometry [Papalambros, 2000]. These can be utilized to simulate system behavior and optimize parameter selections through the evaluation of performance criteria. Other principles guide the selection of final dimensions, form, and parameters. These include design considerations for assembly, manufacturing, failure and reliability, robustness, human factors, and quality [Ullman, 1992][Stoll, 1997][Dieter, 2000][Otto, 2001].

2.1.5 Design Process Scope

Our work concentrates on the conceptual and embodiment stages, utilizing FBS modeling to move from an abstract functional concept to a preliminary system layout with rough approximations of form (Figure 2.2). As this transition occurs, feasibility evaluations can be performed at different complexity levels as design information matures. Specifically, at the conceptual stage, a first pass of feasibility for a workcell utilizing certain general material handlers and components can be determined utilizing scale values (and quantitative values if known) for task compatibility measures.

Even though the concepts associate components with sub-functions, the concepts are too rough for performance evaluations without any layout information. Once preliminary layout generation occurs in embodiment design, enough information is available to make approximations for performance metrics that will be more accurately captured in detailed design. These evaluations further eliminate poor design concepts and

help compare different design architectures. Our goal is to develop computational techniques that allow for the automation of these design tasks.

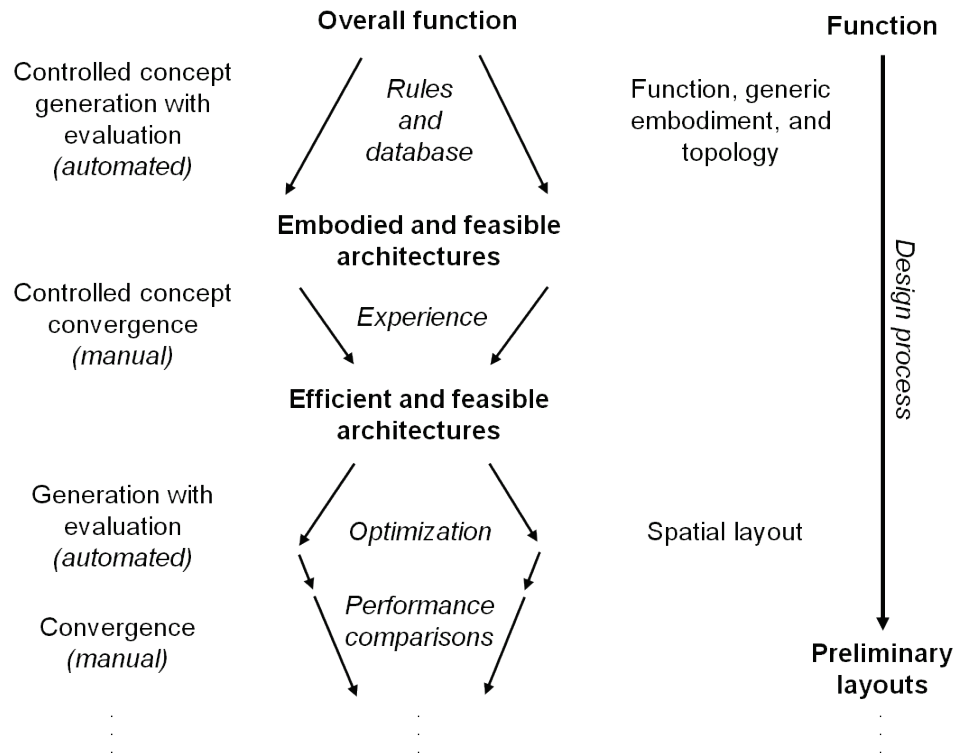


Figure 2.2: The portion of the design process this work addresses and the automated aspects.

2.2 AUTOMATED DESIGN

Much research has been performed on the development of computational tools to automate all or certain parts of the systematic design process described above. A systematic approach is needed to automate the design process (i.e. teach a computer how to design). Automated (or computational) design synthesis (ADS) consists of four generic activities (Figure 2.3):

- (1) *representation* [Campbell, 2003a] or *investigate* [Shea, 2003],
- (2) *generation* [Campbell, 2003a][Shea, 2003],
- (3) *evaluation* [Campbell, 2003a][Shea, 2003], and
- (4) *guidance* [Campbell, 2003a] or *mediate* [Shea, 2003].

Application of one or more of these activities is necessary for automating any part of the design process.

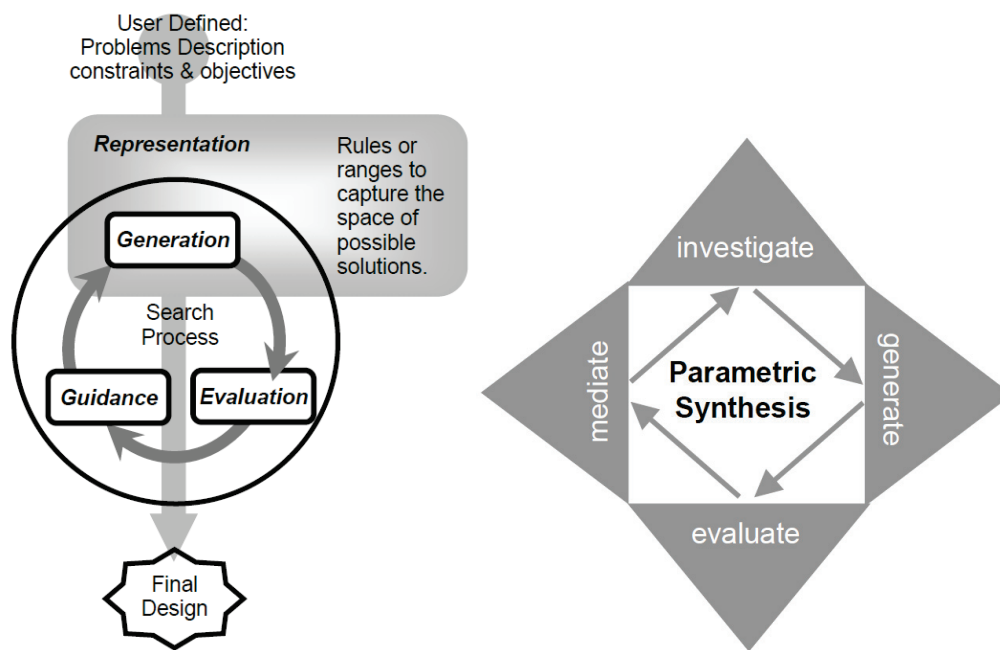


Figure 2.3: Flowcharts for computational synthesis stages from Campbell [2003a] (left) and Shea [2003] (right).

2.2.1 Representation

The representation of design concepts determines how designs are formulated and establishes the domain of designs that can be generated. Synthesis is often viewed as the creation of a form that meets requirements [Cagan, 2005]. Commonly, this form results

from an initial functional description, and the transformation is facilitated by an underlying representation. Function structure generation often consists of individual “black-box” units connected by their inputs and outputs. The Configuration Flow Graph [Kurtoglu, 2008] represents design topology as a graph where generic components are represented by nodes and flows between nodes are represented as arcs. Thus, design concepts can be described at various level of abstraction.

A design can also be represented as a set of design variables. These variables may be directly modified during an optimization process or first coded as a bit string chromosome as in the case of genetic algorithms. Grammars present a different representation paradigm and can describe a design concept by a sequence of operations [Campbell, 2003a]. Similarly, a design could be viewed as a sequence of dependent decisions represented by a search tree [Cagan, 2005].

2.2.2 Generation

The design representation or language heavily affects the mechanism for concept generation. The generative mechanism must be capable of producing both existing and new designs, complex and meaningful designs, and accurately represent the design space [Shea, 2003]. The generation method may be naïve or knowledge-based [Cagan, 2005]. Optimization techniques also provide useful means for generation/search. If gradient information is available, it can suggest search directions. Simulated annealing uses stochastic means to find a neighboring concept from a perturbation of an existing one [Kirkpatrick, 1983]. Genetic algorithms utilize random mutations and crossover operations to find new design states [Renner, 2003]. Generative grammars (Section 2.3.2) apply rules on representations such as graphs to transform designs to new states [Stiny,

1980][Shea, 2003][Sridharan, 2005]. Tree traversal from an initial point using a search algorithm such as breadth-first or A* can produce designs [Campbell, 2003a]. Design generation may also employ interacting agents using a variety of collaboration techniques and/or a knowledge-base to emulate human abilities [Campbell, 2003b].

2.2.3 Evaluation

Evaluation methods are needed to compare designs and determine their performance as in systematic design. The goal is to provide information that accurately captures the quality of generated designs and provides feedback for the guidance stage. Multiple-objectives and their tradeoffs may need to be addressed, requiring selection of appropriate evaluation methods. Computational analysis and complex simulations cannot typically be implemented in this stage. One of the major problems is the potential for the run time to escalate. It is not reasonable to execute a single analysis run that takes on the order of an hour for hundreds (or thousands) of concepts. Other implementation issues include simulation difficulties such as detailed pre-processing, data post-processing, and handling failures and errors [Cagan, 2005]. Any implementation requires a tradeoff between model complexity or design representation realism and analysis time [Shea, 2003]. One recommendation is utilizing quick evaluation heuristics in early searches to steer toward productive areas of the design space [Cagan, 2005]. A more complex evaluation could be made later in the search process or when fewer alternatives exist.

2.2.4 Guidance

The guidance stage utilizes evaluations to improve future generated design solutions. Two approaches for guidance are a real time iteration strategy devised for

subsequent iterations and a long-term strategy where designers learn from the execution of the algorithm to improve future design tasks [Cagan, 2005]. Reasoning mechanisms for evaluations include optimization and search, case-based reasoning, and machine learning [Shea, 2003]. The utilization of optimization principles involves methods for selecting the current “best” design or using information about “useful” designs. For example, in genetic algorithms, “fitness” measures are used to select which design is carried to the next generation. In simulated annealing, the Metropolis criteria [Kirkpatrick, 1983] is utilized early in the search process to accept inferior solutions that help the algorithm avoid local minima, while narrowing in on the best solution later in the search. Knowledge and experience can also be incorporated into guidance techniques to learn from past design generations and build heuristics. Combinations of techniques are also useful, such as mixing grammar and optimization approaches [Cagan, 1993][Shea, 2003]. In general, “stochastic processes are often combined with generative representations that are less knowledge intensive while deterministic search processes combine better with more knowledge-based representations” [Shea, 2003]. Guidance is often hard to separate from generation, especially in optimization approaches [Cagan, 2005], but understanding the dependence of implementation choices for each stage is crucial for developing an efficient and effective automated design technique.

2.2.5 Automated Design Scope

A key factor in the development of our ADS technique is the ability to incorporate the function-behavior-structure framework. To transition from a functional description to one containing structures, we must carefully select our design representation and generation methods. In particular, much domain knowledge is available for concept

generation and should be utilized. More complex evaluation automation is not required until the construction of preliminary layouts because concepts are very abstract before this stage. During layout design, there are a number of decision variables to address (particularly component locations), and this process easily lends itself to optimization with inherent guidance abilities. Thus, our approach will incorporate all aspects of ADS to some extent.

2.3 COMPUTATIONAL TOOLS

There has been extensive research to develop computational tools to automate various parts of the design process using techniques such as those described above. In particular, less tools have been created for automating the conceptual design stage [Strawbridge, 2002][Kurtoglu, 2008] which is more abstract and open-ended in nature. Key elements that have been automated are function structure generation, configuration (topologic) design, and component selection [Kurtoglu, 2008]. The application areas are also broad: electromechanical design, shape generation, geometric packing, construction/structural design, manufacturing, system layout, etc. We will give an overview of some of these tools as a lead in to the selection of the computational techniques utilized for automatically generating solutions to our design problem.

2.3.1 Matrix Formulations

Matrices can help store design knowledge and customer importance and aid in concept generation and evaluation. Many matrix-based computational techniques rely on the incorporation of previously determined design knowledge or computations to guide the development of future designs.

McAdams [2002] developed a procedure to relate customer needs to sub-functions provided by products to generate a function-product matrix listing the functional importance of each function for each product. This matrix can be utilized to compute a product similarity metric that can guide development of analogous designs.

Bryant [2005] utilized a function-function “connectivity matrix”, a function-component matrix (FCM), and a component-component compatibility matrix or “design structure matrix” (DSM) to store relationships and compatibility information for concept generation. Operations performed with these matrices generates an aggregated matrix that can be traced to reveal plausible component chains for a given function chain. Design needs can also be incorporated into the FCM and DSM to rank generated concepts.

Strawbridge [2002] utilized knowledge extracted from disassembly and functional modeling of numerous products to produce a Chi matrix showing how often a given component is utilized to fulfill a particular sub-function. Multiplying the Chi matrix by a filter matrix listing the functions of interest for a particular design gives a morphological matrix where the highest entry in a row corresponds to the suggested component for use.

Olvander [2009] created a quantified morphological matrix that utilizes mathematical models of the solution principles for a given sub-function. The characteristics of one concept variant are obtained by aggregating the characteristics of each solution principle in the concept. An optimization problem can then be constructed to determine the optimal concept variant based on design objectives.

2.3.2 Grammars

Grammar approaches address design representation and provide rule-based design generation. Design grammars specify the valid operations for how components can be

arranged into an acceptable configuration to form a device [Rinderle, 1991] and allow a designer to construct a set of rules that captures his/her knowledge about a certain type of artifact [Kurtoglu, 2005b]. The generative nature of rule application in grammars makes them a useful tool for design generation.

Shape grammars find particular application for design geometry and form (2-D and 3-D) [Stiny, 1980][Agarwal, 2000]. A shape grammar is a set of shape rules that when applied in a step by step fashion on a starting shape generate a set, or language, of designs [Kurtoglu, 2005b]. Shape grammars have been applied to generate complex two and three-dimensional spatial designs [Stiny, 1972][Cagan, 1993][Wang, 2002].

Graph grammars, an offshoot of shape grammars, have also been developed for design generation [Sridharan, 2005][Kurtoglu, 2008][Helms, 2009]. A design is composed of discrete elements that are interconnected, so a graph representation is a natural fit [Helms, 2009]. Graph grammars consist of rules for manipulating nodes and arcs within a graph, creating a formal language for generating and updating complex designs from an initial graph-based representation [Kurtoglu, 2008]. These rules encapsulate a set of valid operations that transform the design into a new state in order to explore different design alternatives and incrementally reach a desired solution.

Graph grammars can construct function structures by utilizing rules to transform graphs where nodes represent functions and arcs represent flows [Sridharan, 2005]. Additionally, a separate graph grammar can be used to transform a function structure into a set of configuration-based graphs called Configuration Flow Graphs [Kurtoglu, 2008]. Schmidt [1998] incorporates grammars into a computational synthesis tool that can perform conceptual design, configuration design, and component selection for drill power trains. Shea [2003] developed a structural grammar that creates classes of discrete truss structures and a parallel mechanical grammar for watch and clock design. Attribute

grammars can extend graph grammars by associating with each symbol in the grammar a set of attributes and every production rule a set of attribute relations [Rinderle, 1991]. This helps represent design requirements and parameters and give meaning and value to constructs produced by the grammar.

2.3.3 Knowledge-Based Systems

Knowledge-based or expert systems are used to emulate the design ability of a human designer for all or various parts of the design process. This requires the knowledge engineering described in Section 2.1.2.3. The main modules of a knowledge-based system (KBS) are the knowledge-base, inference engine, user interface, and, depending on the application, an explanation and/or knowledge acquisition facility (Figure 2.4). The knowledge-base stores information and data about the design domain. Many design applications store knowledge in the form of rules that specify relations, directives, strategies, or heuristics [Negnevitsky, 2005]. Other knowledge representation structures include frames and semantic nets. Knowledge-based systems will be further discussed in Chapter 4.

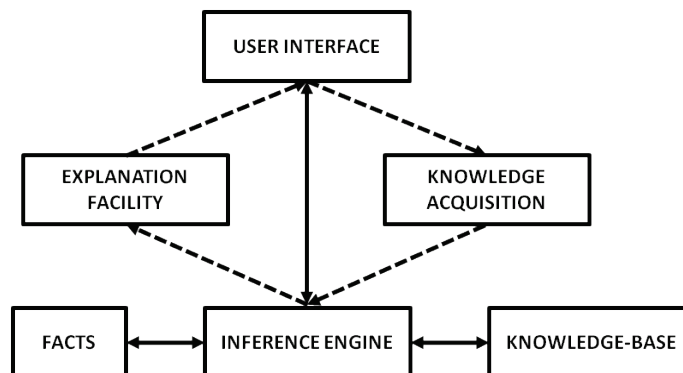


Figure 2.4: The general components of a knowledge-based system.

Knowledge-based systems (KBS) have been extensively applied to design. In the early design stages, they have been used for conceptual and functional design [Theodoracatos, 1994][Zhang, 2002][Woldemichael, 2011][Chen, 2012]. In the realm of manufacturing, they have been applied to design for manufacturing and assembly and machine and tool selection [Kusiak, 1988][Pham, 1988][Venkatachalam, 1993][Zha, 2001]. From these examples, knowledge-based systems can encompass design knowledge and techniques for executing common design tasks near the same level of a human expert.

2.3.4 Agent-Based Design

Agent-based design systems also utilize design domain knowledge to mimic human designers' execution of various design tasks. In these approaches, "design agents have specialized knowledge about the problem domain in which they operate" [Lander, 1997]. For instance, the agents of A-Design "encapsulate knowledge-driven strategies for solving open-ended problems that, through their collaboration, achieve a specified design goal" [Campbell, 2003b]. There are agents with knowledge to determine how components should be connected, which components should fulfill design configurations, how current designs can be improved for future iterations, and how well designs are meeting design requirements. In agent-based design, there is typically an aspect of agent learning that occurs. In A-Design, agents incorporate user input and information from the quality and commonalities of generated designs to guide future designs. The agents in the work of Grecu [2000] utilize empirical information to determine expected events or values from a set of circumstances or conditions.

2.3.5 Catalog Design and Databases

Based on much of the work described, catalogs and databases are particularly useful in their ability to store domain knowledge and interface with computational design tools. Selection design commonly refers to choosing components from a catalog that lists similar components [Ullman, 1992][Dieter, 2000]. Typically, a layout or configuration of generic components must first be defined [Carlson-Skalak, 1998]. Detailed design has benefited greatly from attempts to encode accessible databases within computational tools [Dym, 1994]. Ward [1989] developed a mechanical design “compiler” that given a schematic, specifications, and utility function for a design can determine the optimal catalog numbers for the components in the schematic. Harmer [1998] described a component selection technique using selection charts and generated component property profiles. Potential components are identified by graphically comparing the desired property profile with profiles of components in an assembled database. In A-Design [Campbell, 2003b], there are agents that determine how generic components should be connected and instantiation-agents (“I-agents”) that find specific components from a catalog to fulfill configurations.

Databases and catalogs may be present at levels of abstraction other than components and parts such as generic components or general working principles. Roth [1981] lists three kinds of catalogs: for similar elements (object catalogs), for working rules (operation catalogs), and for the solution of specified sub-functions (solution catalogs). Thus, catalogs can store function, behavior, form, and solution information. Kurtoglu [2005] describes a design repository constructed from analysis and disassembly of existing consumer products that aids functional design synthesis. Zhang [2002] incorporates a behavior knowledge-base into an expert system for functional design. Hubka [1987] also presented catalogs of physical effects or working principles. Using

traditional concept generation techniques such as those identified in Section 2.1.2.4, Woldemichael [2011] developed an alternative concepts database to query for solutions to functions in a morphology chart.

2.3.6 Engineering Optimization

As previously described, optimization principles play a key role in automated design synthesis techniques. Optimization concepts can be applied throughout the whole design process and are incorporated in many of the tools described above. In particular, they can guide the search of the design space, perform single or multi-objective evaluations, and provide guidance for future design development. Additional applications include using simulated annealing mixed with grammars for configuration design [Schmidt, 1998] and optimal shape generation [Cagan, 1993] and genetic algorithms for conceptual design, topology and shape optimization, and parametric design [Renner, 2003][Roy, 2008]. Evolutionary algorithms have also been used to perform catalog design [Brown, 1993][Carlson-Skalak, 1998].

2.4 BACKGROUND SUMMARY

There are many techniques at our disposal for creating a computational design tool and not all of them can be covered here. One commonality of techniques is the ability to incorporate design knowledge and apply it effectively at the desired abstraction level to generation and evaluation procedures. Technique selection relies on the nature of our design problem, design representation, work scope, and work objectives. With these computational tools and our focus area of the design process in mind, we next present our design approach.

2.5 HIGH-LEVEL DESIGN APPROACH

The previous sections presented a background into the design process, the general approach to automating design tasks, and current research in the field of automated design. Based on this information, this section gives the general approach to the design problem described in Chapter 1. We seek to emulate how a human design team would utilize domain knowledge and heuristics to design a glovebox processing system in the nuclear domain. A more knowledge-based approach helps avoid previous design pitfalls and more efficiently steer toward an acceptable design.

2.5.1 Design Needs

We focus on two main parts before detailed design: conceptualizing feasible hybrid workcell concepts and the initial layout of selected workcells (as seen in Figure 2.2). These tasks not only logically flow from the structure of the design process, but are necessary to address important aspects of design concepts generated in our problem domain. Selection of operators (i.e. material handlers), components and apparatuses, and the system layout influence design feasibility and performance (Table 2.1). The handled materials are set by the process and not considered variable, but particularly influence the selection of operators and components.

The incorporation of design knowledge is important for two main reasons. First, during the conceptual design stage, multi-domain knowledge must be communicated between disciplines to avoid infeasible concepts. Second, a wealth of solution knowledge exists regarding typical manufacturing system types and successful designs implemented in the nuclear domain. Since one design goal is to incorporate more automation and robotics into generated designs, modifications to existing manual designs are very useful.

Thus, prior knowledge steers designs away from infeasible combinations and more quickly directs concepts toward feasible variants.

Concern \ Factor	Materials	Operators	Components	Layout
Feasibility				
- <i>Task/Function</i>	X	X	X	
- <i>Workcell</i>		X	X	X
Performance				
- ALARA				
- <i>Time</i>	X	X	X	X
- <i>Distance</i>	X	X		X
- <i>Shielding</i>	X	X	X	X
- Ergonomics & Safety	X	X	X	X
- Manufacturing				
- <i>Time</i>	X	X	X	X
- <i>Quality</i>	X	X	X	

Table 2.1: The main factors that influence design feasibility and performance. An abbreviated list of performance metrics is presented.

2.5.2 Computational Technique Overview

Our approach addresses the above needs, presenting a computational framework for automating design under these conditions. FBS modeling facilitates the transition from an overall functional process description to a system design concept capable of executing that process. The main mechanism for dictating this development is a rule-based knowledge-based system. This will allow us to capture and update process-based knowledge and progress design concepts in a structured manner. We will also take advantage of the linear nature of our processes when developing our function structure. generation technique and design representation.

In order to establish a consistent means of communication between computer and human, we will utilize a common language for describing functions and flows. Other general knowledge related to prior designs (ex: general component, material, and operator types) can be stored in a database that interfaces with the knowledge-base rules. This and the common language can be stored in structures such as frames/classes.

The KBS concept generation technique will provide initial filtering of feasible concepts when exact parameters and geometric dimensions are still unknown. Developing preliminary design layouts using rough spatial and geometric information addresses workcell concept feasibility more accurately and allows for approximate performance evaluations of important criteria. Thus, determining feasible workcell layouts is an important part of concept development.

The main layout task is to determine good locations for the components and operators within a workcell. There are many considerations for component placement such as collisions among components, operator reachability, and collision-free task trajectories. Beyond these feasibility assessments, optimization can help search for the best layout based on objectives such as dose and processing time minimization.

The optimization process will visit on the order of thousands of designs and thus approximations are needed to speed up calculations. Components can be modeled by simple geometric shapes such as rectangles and cylinders to detect collisions and trajectories can be simplified to a sequence of linear segments. We must also simplify dose calculations and make time approximations for various tasks based on the material handler. A weighted sum approach allows for multiple objectives to be addressed.

We construct an extended pattern search algorithm to search for component locations. This hybrid deterministic and stochastic search algorithm can handle our optimization problem formulation and the difficulties in searching the design space. Also,

the algorithm design representation is flexible to handle implementation extensions, if desired. This could include generative rules that allow concepts to be further modified (i.e. changes in the numbers of design variables) during the optimization process.

This is a knowledge-based approach, so we will present additional information and techniques for the implementation of different ADS modules as needed. This is particularly necessary for selecting metrics for robotic and human task feasibility, determining workcell layout, and formulating a radiation dose approximation for evaluation.

2.5.3 Summary

This implementation framework is modeled after how an experienced design team may approach this design problem. The rules work similarly to a process expert recalling information about major system functions and how the process should flow from beginning to end. The component database acts similar to human memory for recalling structures capable of performing functions. The optimization algorithm improves the common trial-and-error technique for workcell layout consisting of numerous manual cycles of layout modification and evaluation. In the end, automated design is utilized to emulate how engineers develop glovebox systems in the nuclear domain. The main design tasks include establishing system functions, selecting components and labor, developing operating procedures, and finding a feasible (and perhaps “best”) layout.

In the following chapters, we will present our research in two stages. First, we will describe the implementation of our concept generation technique (Chapters 3 and 4) and apply it to manufacturing processes similar to those conducted at LANL (Chapter 5). The second stage addresses workcell layout optimization for conceptual design (Chapters

6 and 7). The optimization algorithm (Chapter 8) will be used to optimize workcell designs from our concept generation technique (Chapter 9).

Chapter 3: Concept Generation Modeling

Our concept generation technique has two main tasks: determining the structure of FBS modeling (this chapter) and the mechanism for performing it (Chapter 4). The first task depends on how we define the systems we want to design and the considerations for executing FBS modeling for this design problem. The proposed FBS modeling mechanism based on the system formulation and available design knowledge is a knowledge-based system (KBS).

3.1 SYSTEM DEFINITION

Two definitions of system are “the entire combination of hardware, information, and people necessary to accomplish some specific mission” [Dieter, 2000] or “a group of components that work together for a specified purpose” [Sage, 2000]. This implies that systems have specific tasks or functions in order to achieve their purpose. What belongs to a system is determined by its defined boundary and the inputs and outputs that cross the boundary to connect with the surrounding environment [Pahl, 2007].

The same FBS framework commonly used for product design can be utilized for workcell design. In both cases, each can be seen as a system, although at different abstraction levels. The emphasis is on the overall function of the system, its inputs and outputs, and the system’s decomposition into sub-units to make design more manageable. This is not “systems engineering” [Sage, 2000] in the sense of the discipline where the full life cycle, management considerations, resource allocations, etc, are all handled.

3.2 FUNCTION AND PROCESS

Specifically, we deal with manufacturing systems consisting of “the arrangement and operation of elements (machines, tools, material, people and information) to produce a value-added physical, informational or service product” [Suh, 1998]. Our inputs are typically raw materials (many nuclear) that through the functions of the system are converted to a material product with a more defined form. There are specific chemical, mechanical, etc, sub-processes that must be performed and included in our generated designs. These are only addressed to find a generic apparatus or machine that can perform them. We are not modifying the essential sub-process but rather determining how to carry them out or link them together using the operators and components at our disposal. Most of our design flexibility comes from the ability to select different compatible apparatuses for a specific sub-process and choose among multiple material handlers to move materials between these apparatuses. Thus, the system boundary encompasses all the components, apparatuses, and material handlers (human or robotic) to execute the needed sub-processes and related material transfer tasks.

3.2.1 Process Planning

As described, many operations stem from the components used for processing and the transfer tasks needed to move materials between locations. The sequence of operations is also critical to performing the process correctly. Therefore, when conceptualizing a manufacturing workcell, we must also consider the task plan or aspects of *process planning*. From a technical viewpoint, “process planning can be defined as an act of preparing detailed processing documentation for the manufacture of a piece part or assembly” [Kiritsis, 1995]. Process planning considers several design tasks including operation sequences and machine, tool, and labor selection. Computer-Aided Process

Planning (CAPP) [Kiritsis, 1995][Marri, 1998] is typically performed at the detailed design level using a CAD model. CAPP computational tools commonly address a specific machining sequence for a part or product incorporating knowledge about part form, features, materials, requirements (tolerances), etc.

There are two basic approaches to automated process planning: the variant approach and the generative approach [Kusiak, 1991][Brown, 1997]. The variant approach utilizes the Group Technology (GT) concept to retrieve a plan from a database for a new part based on a code calculated from its attributes and features. Modifications can be made to the plan as necessary. The generative approach “attempts to create a completely new plan for each part, using knowledge of manufacturing operations, process capabilities, and the part description” [Brown, 1997].

Feng [2000] addresses process planning for conceptual design rather than for detailed design. Computational tools for conceptual design and manufacturing are not as prevalent as detailed techniques that utilize CAD. Often, these tools “do not address many aspects of conceptual design, including functional decomposition and mapping from functions to the designed product” [Feng, 2000]. Mukherjee [1997] utilized a GT-based coding scheme for representing function information and developed sketching abstractions to represent critical part geometry information to help link pure functional and pure geometric representations. Salehi [2009] utilized a genetic algorithm to generate feasible part manufacturing sequences using order constraints. Then optimization determines the optimal sequence and machine and cutting tool selection. Many more examples exist for detailed CAPP, including generative approaches [Marri, 1998] and expert systems [Kiritsis, 1995].

3.2.2 Robotic Task Planning

A subset of general process planning is robotic task planning for manufacturing. Automatic generation of most robotic task plans also centers around the part or product being manufactured [Gottschlich, 1994][Mosemann, 2001]. The goal is to automatically synthesize a detailed robotic operation plan from a high-level product description [Gottschlich, 1994]. The high-level plan is typically composed of symbolic operations such as “put A on B”, “weld C on D”, and “insert E in F” [Rocha, 1997]. Then the user generates a program for controlling a robot to perform those operations.

Assembly plans frequently utilize AND/OR graphs which can generate all feasible assemblies [Rocha, 1995]. Petri nets are another representation commonly employed. A Petri net graph consists of circle nodes that represent a *place* and bar nodes that represent *transitions* [Cao, 1991]. A directed arc linking circle and bar nodes indicates the relation between them. For assembly or disassembly, places can represent contact states in the assembly and transitions can represent the gaining or losing of contact [Rosell, 2003].

Cao [1991] mapped AND/OR graphs to Petri nets where task sequences were found from the reachability tree of the Petri net. Rosell [2003] used Petri nets to find near-optimal robot assembly sequences taking into account the cost of operations. The technique used a precedence matrix (which parts must be removed before others) and neighboring contact matrices (showing edge-edge contact between two parts). Mosemann [2001] broke down the fine motion planning of assembly tasks into skill primitives with start and goal conditions.

3.2.3 Process Planning Influence

We are concerned with conceptualizing a system as a whole without particular emphasis on the specific operations needed to physically manufacture a part. Thus, our task plans will not correspond to detailed manufacturing sequences describing the function that an apparatus/component performs. Working at the system and conceptual levels, most knowledge will be generic, lacking the explicit and complete technical information needed to produce a finished part [Mukherjee, 1998]. Approximate specifications for mass, shape, and trajectory may be needed for rough feasibility assessments, but specifications such as tolerances and precise machining sequences and trajectories are not utilized.

The basic principles of process planning are still relevant and should be incorporated into our concept generation technique. For this particular design problem, we are most concerned with the tasks needed to ready an object for machine and tool use. Important processing considerations include preserving the correct precedence of operations and making the proper selections/embodiments for functions, both machine and labor. In particular, the selection of system components influences how some tasks must be performed and/or what additional system tasks are necessary. For example, a robot and human may execute the same task using different means or apparatuses may have different sub-tasks for a worker to operate them. In such cases, the inputs and outputs of the sub-process/function are the same, but the supporting tasks differ. Thus, as the overall system function is decomposed, some functions must be embodied (by components or operators) to reveal additional sub-tasks that are necessary in the function structure. Such knowledge must be stored or obtained during the design process. Other aspects of process planning such as resource selection and scheduling are also not

addressed in our design process. Process planning and its effect on function structure generation will be described more in-depth in Section 3.6.

3.3 DESIGN REPRESENTATION

Our design concepts are defined by the lowest level system functions in the function structure and the components and operators used to embody them. Given the order built into the process, a design's function structure can be represented as an ordered sequence of functions. This is essentially a directed graph where each node represents a function, and an arc represents the output flows of one node to the input flows of another. That is, the connections between functions show the flow of material and components used in the process and material transfer tasks. Since function embodiment is necessary to expand the function structure, a completed design is a function structure with components (at some level of abstraction) assigned to each sub-function. This design representation is similar to a Configuration Flow Graph [Kurtoglu, 2008], and its generation is a single step instead of a two step process (from overall function to function structure, then function structure to Configuration Flow Graph). Both representations reveal information about the connectivity of system components.

3.4 COMMON LANGUAGE

Throughout all KBS modules is a common language of functions and flows so the computer has a language for FBS modeling to generate designs and communicate with the designer. The definition of function is critical to the success of implementing an artificial intelligence technique [Erden, 2008]. Based on our system modeling, we adhere to the device-centered view of function where a function is given as a relationship of

inputs and outputs. Although our systems execute processes, we do not utilize the process-centered view since system components are not passive but rather have specific sub-functions to execute. The device-centered representation allows the tracking/modeling of flow movement between components and actions performed by components on materials and parts. Since we are not focused on the process and the device-centered view can sufficiently contain all the information we need, we do not use other process modeling techniques such as IDEF0 or its other variants [IDEF, 2013]. This modeling language contains more than what we need and cannot always ensure uniform understanding from all users [Nagel, 2006].

Nagel [2006] utilizes a process modeling methodology that builds time dependence into traditional functional modeling techniques. In our formulation, time dependent information is captured by the order of functions in the function structure. This is in the form of precedence information and not the exact timing of operations on a common scale. This means a function cannot be executed until all of its input flows have been resolved, that is, all the prior functions utilizing those flows have been executed. Sometimes functions are independent and can be executed in any order. In these cases, a particular order is pre-selected and defined in the rules.

3.4.1 Functional Basis

Several researchers have worked to define functional terms for mechanical design [Altshuller, 1984][Hundal, 1990][Hirtz, 2001][Pahl, 2007]. Of these, the Functional Basis [Hirtz, 2001] has found much application [Sridharan, 2005][Bryant, 2005][Helms, 2009]. This basis (Figure 3.1) utilizes a function representation based on relationships of inputs and outputs with the typical “verb + noun” modeling. The term “basis” was adopted to

infer that this taxonomy represents the minimum terms necessary to describe the design of engineering artifacts, products, and systems. The Functional Basis can provide clearer task understanding than IDEF0 [Nagel, 2006].

We utilize the Functional Basis as the baseline to develop a vocabulary describing functions (Figure 3.2) and flows (Figure 3.4) for manufacturing system processing. We also seek a type of “basis” or minimal collection of terms to describe the design domain. In our formulation, emphasis is placed on incorporating manufacturing and material handling functions which can also be viewed similarly to sub-processes. A major difference between Hirtz’s formulation and ours is the detail level of our designs. Referencing Figure 1.2, our level of detail typically stops around the *Component* level, whereas Hirtz addresses electromechanical design incorporating *Subcomponents* and *Parts*. Thus, the scope of our work starts at a *System* level rather than a *Component* level.

In our language, the *Primary* class designations are retained. Terms from the *Secondary* and *Tertiary* classes are combined or removed based on their applicability to manufacturing or the desired functional resolution for our conceptual designs. In particular, many signal/energy flow processing functions are removed since our designs do not typically address this level of detail. Also, in place of *Secondary* and *Tertiary* designations, the more abstract high-level functions are termed “non-terminal” functions and the lowest-level functions are called “terminal” functions [Baldwin, 1995]. Non-terminal functions decompose into sequences of other non-terminal and/or terminal functions. This lends itself to a hierarchical structure of terms similar to the Functional Basis. The non-terminal functions in the first column represent typical material handling functions (added by the KBS during concept generation), and the second column lists specific manufacturing functions (added by the user and extendable). The terminal

functions typically represent the point at which a tool or component invokes its function and the most concrete processing information for our concepts.

<i>Class (Primary)</i>	<i>Secondary</i>	<i>Tertiary</i>
Branch	Separate	
		Divide
		Extract
		Remove
Channel	Distribute	
	Import	
	Export	
	Transfer	
Connect		Transport
		Transmit
	Guide	Translate
		Rotate
Control		Allow DOF
	Couple	
		Join
		Link
Magnitude	Mix	
	Actuate	
	Regulate	Increase
		Decrease
Change		
		Increment
		Decrement
		Shape
Stop		Condition
	Stop	
		Prevent
		Inhibit
Convert	Convert	
Provision	Store	
		Contain
		Collect
	Supply	
Signal	Sense	
		Detect
		Measure
	Indicate	
Support		Track
		Display
	Process	
	Stabilize	
	Secure	
	Position	

<i>Class (Primary)</i>	<i>Secondary</i>	<i>Tertiary</i>
Material	Human	
	Gas	
	Liquid	
	Solid	Object
		Particulate
		Composite
	Plasma	
	Mixture	Gas-gas
		Liquid-liquid
		Solid-solid
Signal		Solid-Liquid
		Liquid-Gas
		Solid-Gas
		Solid-Liquid-Gas
		Colloidal
	Status	Auditory
		Olfactory
		Tactile
		Taste
		Visual
Energy	Control	Analog
		Discrete
	Human	
	Acoustic	
	Biological	
	Chemical	
	Electrical	
	Electromagnetic	Optical
		Solar
	Hydraulic	
Energy	Magnetic	
	Mechanical	Rotational
		Translational
	Pneumatic	
	Radioactive / Nuclear	
	Thermal	

Figure 3.1: The function and flow classes of the Functional Basis [Hirtz, 2001].

<i>Class</i>	<i>Non-terminal</i>		<i>Terminal</i>	<i>Correspondents</i>
Branch	Open			
(Separate)	Disconnect			Disassemble, unfasten
	Unload			Unpackage
		Sort		Divide, split
	(Remove)	Cut		Shear
		Drill		Mill
		Lathe		Carve
		Break		Crush, chip
		Grind		Sand
			<i>Separate</i>	Remove
			<i>Release</i>	Ungrasp
Channel	Transport			Transfer, lift
(Move)	Dispense			Pour, funnel, scatter, eject
		(Move)	<i>Move</i>	Move over, follow
			<i>Translate</i>	Slide
			<i>Rotate</i>	Turn, spin
			<i>Agitate</i>	Shake, homogenize
Connect	Close			
(Assemble)	Connect			Assemble, fasten, attach
	Load			Package, insert
		Mix		Combine, add together, stir
	(Assemble)	Weld		Solder
		Rivet		
		Screw		
		Bolt		
		Nail		
		Adhere		Glue, bond
			<i>Join</i>	
			<i>Grasp</i>	Obtain, clutch
Control Magnitude			<i>Activate</i>	Initiate, turn on
(Control)		(Stop)	<i>Disable</i>	Turn off, deactivate
			<i>Shield</i>	Block
Convert	(Shape)	Form		Mold, cast, press
(Change state)		Deform		Bend, flatten, press
	(Treat)	Coat		Finish
		Clean		Rub, sweep, dust, polish, wipe
	(Convert)	Heat		Sinter, melt, evaporate
		Cool		Freeze
		Precipitate		Condense
		Extract		Filter
			<i>Create</i>	Convert, transform, produce
Provision			<i>Store</i>	Keep, contain, collect
(Store)			<i>Supply</i>	Provide
Signal		Survey		Inspect, monitor
(Sense)		Measure		Section, divide out, split
			<i>Detect</i>	Identify, sense, locate, take data
Support	Stabilize			Steady
(Support)			<i>Support</i>	Hold up, elevate

Figure 3.2: The eight function classes and their non-terminal and terminal functions.

3.4.2 Function Categorization

In our work, we incorporate both human and robotic labor into manufacturing processes. Ghosh [1986] described common manufacturing tasks allocated to humans and robots (Figure 3.3). Mital [1994] described similar tasks.

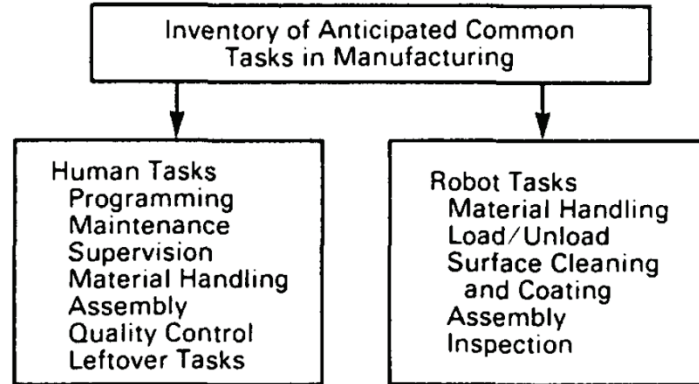


Figure 3.3: Common manufacturing tasks for humans and robots [Ghosh, 1986].

The eight *Primary* classes of the Functional Basis provide a taxonomy for grouping manufacturing functions. A function's class/category is selected based on its input and output relationships and flow types. At their core, most functions fall into a finite number of categories. These correspond to their flow types and the relationship between their inputs and outputs. Some examples can be seen in the generally valid functions described by Pahl [2007]. Often, one input comes in and one modified output comes out, one input is transformed into two, or two inputs are combined into one output. For example, *Branch* functions separate flows, *Connect* functions combine flows, and *Channel* functions pass along the same flow. *Convert* functions could do all of these tasks but are more related to chemical changes or changes in state. Functions dealing with larger numbers of inputs and/or outputs can usually be decomposed into these lower level

representations. For example, loading multiple flows can be decomposed into a *Load* for each flow or breaking apart a combined flow into multiple flows can be seen as a sequence of individual breaks from the main flow.

A function's decomposition can provide a further means of classification once a category is selected. We can group functions that have similar sub-processes or decompositions into the same term. In some cases, somewhat redundant functions are retained if they provide more ease of natural language description. For example, including *Open* even though its decomposition (broadly *Detect*, *Move*, *Grasp*, *Move*, *Release*, *Move*) is nearly identical to *Disconnect* for certain handled flows. Adding symmetry is also useful so one task and its reverse task are both present. Some functional terms may need to be more specific to limit the number of apparatuses that can be assigned to them, while other functions may be more general if they have more flexibility in embodiment. In our case, this is driven by the process. Since a specific process is addressed, simply stating at a high level the need to "Remove" something is not precise enough, as drastic embodiment differences exist between the more specific functions of *Drill* or *Cut*, for instance. However, independent of the processing apparatus selected, a number of general material handling tasks such as *Load* or *Store* could be applicable.

3.4.3 Terminal Function Influence

The decompositions of process functions/tasks are important in developing terminal function terms. Ghosh [1986] divided an assembly task into the subtasks of REACH, SELECT, GRASP, MOVE, POSITION, and ASSEMBLY. Genaidy [1990] reviewed work that used sub-tasks to estimate the time for an operation. The Robot Time and Motion (RTM) method [Nof, 1999] described eleven basic steps to decompose

robotic operations: REACH, MOVE, ORIENT, STOP-ON-ERROR, STOP-ON-FORCE, TOUCH, VISION, GRASP, RELEASE, PROCESS-TIME-DELAY, and TIME-DELAY. The MOST technique [Zandin, 2002] assumes three types of movements [Genaidy, 1990]:

- 1) *General Move Sequence* (free movement of object) with the three phases GET, PUT, and RETURN,
- 2) *Controlled Move Sequence* (maintain contact with surface or object) with three phases GET, MOVE or ACTUATE, and RETURN, and
- 3) *Tool Use Sequence* with the five phases GET OBJECT or TOOL, PLACE OBJECT or TOOL, USE TOOL, ASIDE OBJECT or TOOL, and RETURN.

We are particularly concerned with primitive sequences that describe material handling and processing by humans or robots. Furthermore, the functions executed by machines (or tools) are not as important as the operations needed to manipulate them. Such terms are specific enough for our descriptions because our main emphasis is on what is being moved (or manipulated) and where these tasks occur and not the detailed operational task that occurs at a location.

3.4.4 Language Flows

The flows are also developed in a process-oriented matter. They are essentially divided into objects that are processed and components that do the processing. Processed flows are sub-divided based on their physical properties and material handling considerations. One distinction is made between flows with undefined shapes (raw) and those with more defined forms. The flows' physical states and number and kinds of "ingredients" provide further distinctions. Processing components are sub-divided based

on the typical primary functions they perform during processing operations. *Components* are essentially all flows that embody functions.

Flows/Targets	Primary	Secondary	Correspondents
(Processed)	Material (Raw)	Object	Metal, solid
		Powder	Oxide, granular, particulate, particles
		Liquid	Molten, chemical
		Gas	Fumes, vapor
	Collection (Raw)	Pieces	Same solid, coarse fragments
		Composite	One solid of different materials
		Aggregate	Different loose objects, powders, and/or fragments
		Mixture	Liquid/solid, liquid/liquid, gases
	Item (Formed solid)	Part	Piece
		Connector	Bolt, screw, pin
		Assembly	Combined parts and connectors
(Processing)	Component	Apparatus	Instrument, equipment
		Operator	Human, robot, control system
		Container	Can, crucible, fixture, space, cabinet
		Tool	Gripper, hand, clamp
	Environment	Environment	Surroundings, atmosphere

Figure 3.4: The common manufacturing language flows.

This manufacturing common language is not an exhaustive list of all possible system functions and flows. Further modifications will be evident from extended use. For instance, there are likely more functions that could be grouped into “Remove” and “Assemble”. These are easily added to the basis because the supporting material handling functions already exist and depend on the possible embodiments.

3.5 FUNCTION/TASK COMPATIBILITY

We also need to determine whether human and/or robotic labor is possible for a task. More specifically, we are concerned with feasibility/compatibility and not optimal

selection. Two main sources for developing task compatibility metrics come from the area of task allocation and common robotic task performance metrics.

Task allocation often deals with evaluations of what machines do better than humans and vice versa [Fitts, 1951][Jordan, 1963][Kantowitz, 1987][Mital, 1994][Sheridan, 2000]. These discussions are important for determining the relevant attributes and categories for determining task compatibility for both humans and robotics. Kantowitz [1997] listed some of the functional advantages and disadvantages for men and machines in regard to data sensing, processing, and transmitting and economic properties. Ghosh [1986] listed principles that limit human performance such as repeatability, accuracy, speed of hand movement, reach, lifting capability, and limitations of joints, ligaments, and tendons (ergonomics). Mital [1994] also describes work/task conditions that can force task allocation to either humans or robots.

Analysis of quantitative robotic measures reveals what attributes are important for task completion. Ghosh [1986] identified some robotic performance characteristics including load capacity, reach, repeatability, memory, and degrees of freedom. Bhangale [2004] lists over 80 robot attributes for selection related to physical properties, performance, architecture, and other areas. Performance measures are typically tied to the physical properties of a robot and incorporated into optimization schemes [Gao, 1997][Pholsiri, 2004][Tisius, 2009]. Our generated concepts lack the detailed geometric and parametric information needed to derive these types of metrics. In particular, without kinematic models of robotic manipulator arms and spatial positioning we cannot determine target reachability and dexterity criteria such as the *manipulability measure* [Yoshikawa, 1985]. Additionally, details are lacking for grasp synthesis [Sahbani, 2012] and investigations of the fine motion planning [Lozano-Perez, 1989] required around a task point.

We utilize these sources to develop qualitative and quantitative metrics for human, robot, and apparatus compatibility. Quantitative metrics include *reach*, *payload*, *accuracy*, *repeatability*, and *velocity*, which can also be given qualitative values by establishing numeric scales that relate to “low”, “medium”, and “high” values. Table 3.1 lists some qualitative task metrics and how their values on a scale from 1 to 3 (low to high) are determined. A value of 1 typically represents the minimally acceptable requirement or ability. These values are derived from published work and the author’s experience analyzing typical processes in the nuclear domain and general human and robotic capabilities. These metrics will be utilized in concept generation to identify potential function embodiments. Not all of these requirements are utilized during the concept generation phase, as some are only plausible when a configuration is specified.

	1	2	3
<i>Grasps</i>	Fingertip grasping	Whole hand grasping	Two-handed grasping
<i>Dexterity</i>	Attain at least one hand pose in a sphere about task point	Attain hand poses in 1/2 of a sphere about task point	Attain hand poses in 3/4 or more of a sphere about task point
<i>Motion</i>	Coarse movements in uncluttered areas	Coarse movements near obstacles, requires vision and/or collision detection	Precise movement, needs vision, collision, and/or force detection/feedback
<i>Vision</i>	Identify item or attribute presence	Identify initial item or attribute location accurately and/or differentiate between materials	Track item or attribute location throughout task and/or provide vision feedback for movement
<i>Force</i>	Roughly apply small force or sense large force	Roughly apply medium force, maintain force profile, or detect medium force	Roughly apply large force or precisely maintain force profile or detect small force

Table 3.1: The scale values for qualitative task criteria.

At this stage in the design process, the goal is to eliminate concepts that are discernibly infeasible at first glance which can be done with these types of general categories. For example, distinct differences in grasping needs exist between small and large objects and objects with smoothly curved surfaces and those with protrusions or other complex features. Additionally, the types of apparatuses and components used in a task can help derive constraints. Some tasks must be performed in tight spaces with access limitations while others are much less constrained in clearance and accessibility. This analysis has more bearing on the feasibility of the workcell rather than solely operator task compatibility.

3.6 FUNCTION STRUCTURE GENERATION

The processing nature leads to new considerations for function structure generation. To describe an overall task, we start with a non-terminal function and its associated inputs and outputs. Before this function/task can be performed, some additional tasks may be necessary: loading the correct material into an apparatus, opening an apparatus before loading, etc. Such requirements depend on the function's embodiment and the structure's state. Thus, embodiment is needed to instruct how the development of the function structure should proceed based on processing needs. In the above case, a transformation could be made from the overall non-terminal function to an embodied non-terminal function plus supporting functions. Decomposition can then occur on the supporting functions if no additional tasks are necessary.

Therefore, the ordered nature of the considered processes dictates that some embodiments occur before some functional decomposition as embodiment decisions influence functional needs. That is, embodiment is utilized to capture more functional

information related to processing needs. The functions that are embodied are decomposed enough that structures can be associated with them, so there is no loss of information from their embodiment – they are primitives in the sense of *component* function, but non-primitives in the sense of the *processing* function.

This coincides with our desire to generate creative material handling systems. The process is pre-defined, so the higher-level processes must be specific to capture the process. This means only a select number of components are applicable. However, the main source of creativity comes from the decomposition and embodiment of supporting tasks. Once a specific function is embodied, general material handling functions are added. These can be decomposed and often have multiple possible embodiments. This means a non-terminal or terminal function can be embodied. Non-terminal functions are embodied if they represent specific process-related functions and terminal functions are always embodied.

Therefore, in our formulation, decomposition is performed in the context of the process. Non-terminal functions require decomposition in the traditional sense (process into sub-processes) or that additional functions are added to the function structure. Terminal functions do not require decomposition nor do they indicate the necessity of additional tasks. Since we work at the conceptual level, terminal functions also represent the most specific level of detail for describing instructions in the task sequence. Lacking detailed technical information, the terminal functions are an abstraction of more concrete processes that utilize CAD and precise processing information. For instance, to *Separate* a material using a lathe requires precise fixture placement, part geometries, cutting paths, etc. Terminal function decomposition would reveal the sub-functions occurring in the component itself. For example, the decomposition of a *Break* function embodied by a *Press* contains a terminal function *Separate* which would encompass the physical act of

breaking: the electrical signals traveling in the press, the actuation of the hydraulic cylinder, the movement of the press ram, the transfer of force, etc.

Additionally, some decompositions are abbreviated so multiple operators do not perform tasks that should be assigned to the same operator. For instance, the *Transport* function could be decomposed into *Detect* flow, *Move* to flow, *Grasp* flow, *Move* flow, *Release* flow, and *Move* away from flow. However, in most cases, all but the first task should be executed by the same operator since switching between operators for these tasks does not occur in actual systems. Therefore, little is gained by making this decomposition, so the last five tasks are combined into a single *Move* function that incorporates all aspects of those five functions. This distinction allows a *Camera* to *Detect* an object for a *Robot* to *Move*, but the production rules would have to incorporate a check that disallows a *Camera* to *Detect* when a *Human* is performing the *Move*.

The coupling between function and embodiment results in an iterative function structure generation process that moves between abstraction levels as needed. As more embodiment options are available, more unique function structures/task sequences can be generated. At any time during design construction, the function structure may contain a mix of functions with and without components assigned. A fully embodied function structure is similar to the Configuration Flow Graph developed by Kurtoglu [35].

An example of function structure generation is seen in Figure 3.5. The overall function (top) is to *Break* one solid into three separate solids. First (1), a *Store* function is added and embodied (with *Container 2*) to store the input (*Composite 1*). The overall *Break* function is decomposed into two other *Break* functions (2). The first *Break* is embodied with a *Tool* (3) and has a decomposition that occurs simultaneously. Next, the *Stabilize* function is embodied by a *Human* (4) and the proper decomposition follows (5). Finally, the first *Break* function is decomposed (6) after all necessary processing

information has been added during the previous steps. The generated function structure and task sequence are seen in Figure 3.7. Similar operations are needed to develop the second *Break* produced by the decomposition in (2). The order of function structure development, function additions, function insertion points, and the introduction of new input or output flows are all handled by the KBS rules described in the next sections.

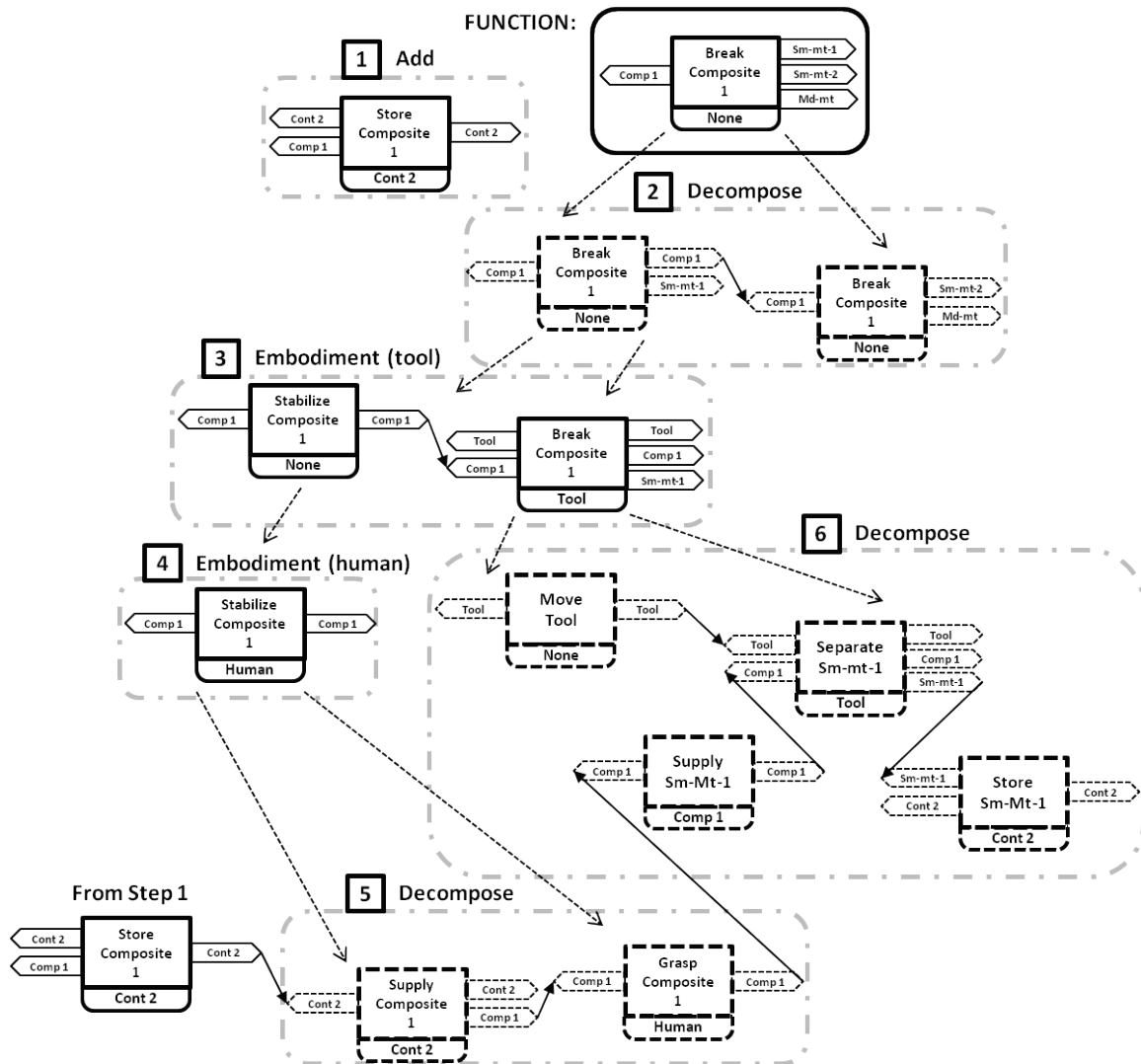


Figure 3.5: An example of function structure generation.

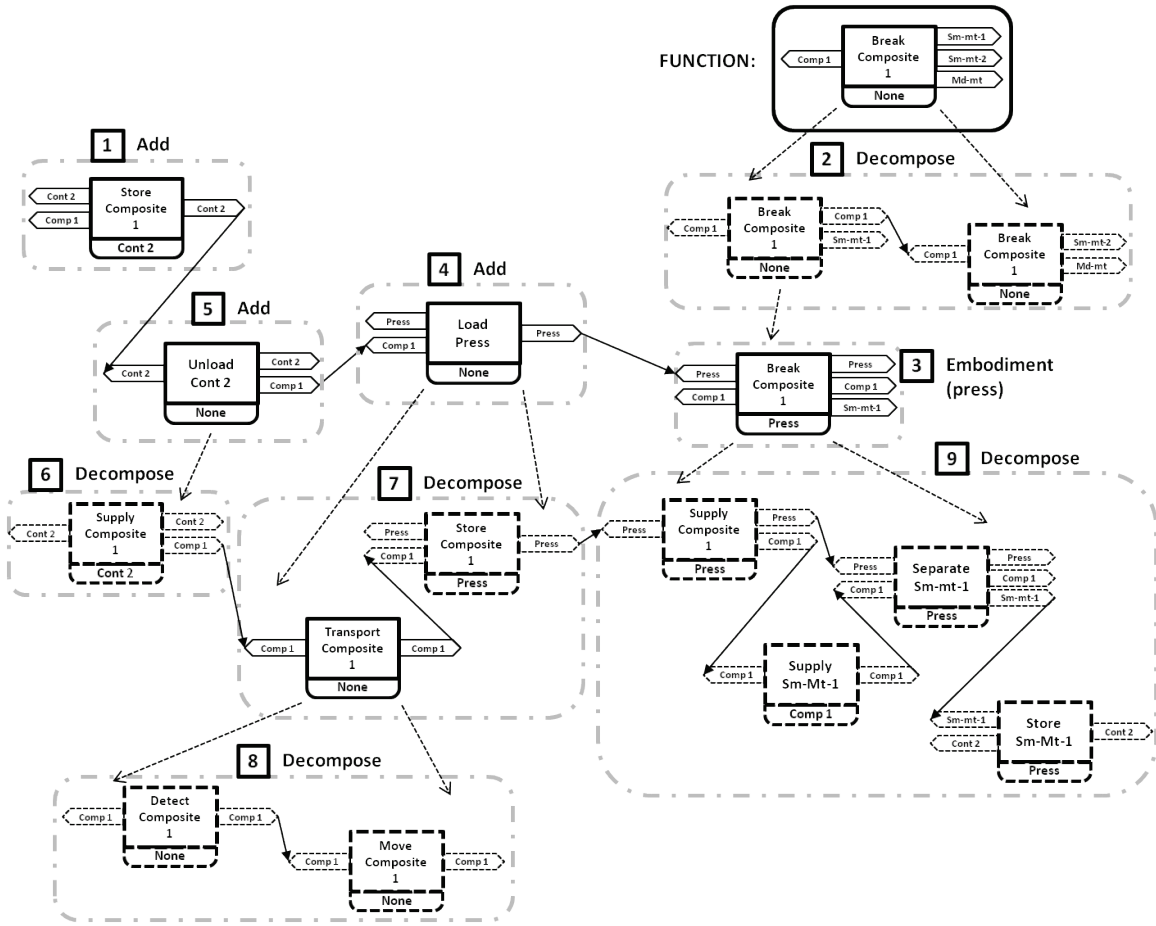


Figure 3.6: An alternate function structure generated if *Press* embodies *Break*.

In Figure 3.6, an alternative function structure is generated when a *Press* is chosen for *Break*. The first two steps are the same then a *Press* is selected to embody *Break* (3). A *Load* is added to load the *Press* with *Composite-1* (4). However, since *Composite-1* is stored in *Container 2*, an *Unload* is added to remove *Composite-1* from *Container 2* (5). This *Unload* is then decomposed (6). If *Container 2* had a lid, an *Open* would have been added before the *Unload*. The *Load* is then decomposed into *Transport* and *Store* (7), and *Transport* is subsequently decomposed into *Detect* and *Move* (8), both terminal functions. With the material handling functions added and decomposed, *Break* is

decomposed into four terminal functions (9). The *Separate* function represents the physical act of the *Press* moving and forcing *Composite-1* to break. The separated material (*Small-mat-1*) is stored in the *Press*. The generated function structure and task plan are shown at the bottom of Figure 3.7.

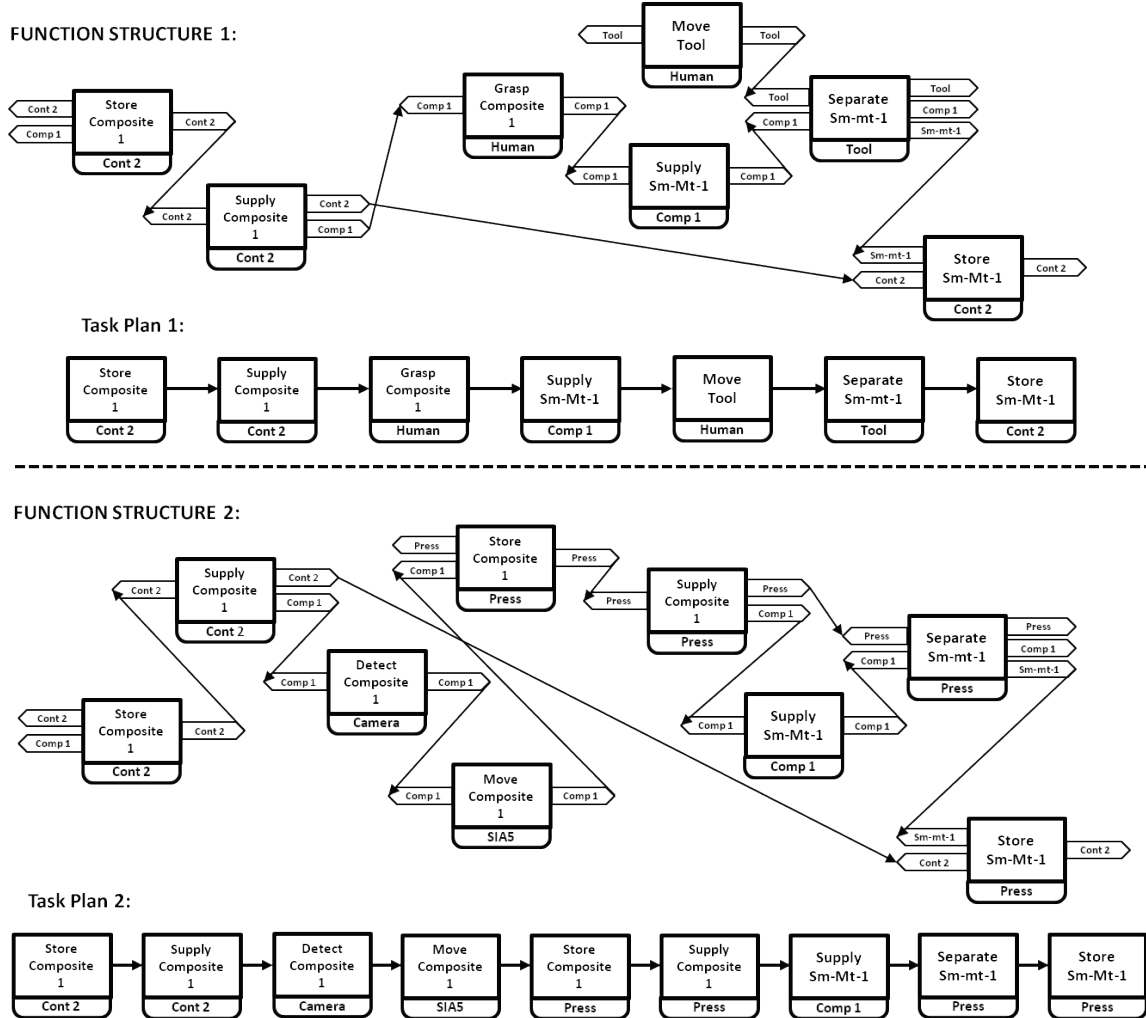


Figure 3.7: The final function structures and task plans of Figure 3.5 and Figure 3.6.

The final (partial) function structures differ not only in the selected components and operators, but in the number and type of functions in the function structure (Figure 3.7). The final embodiments for terminal functions have also been performed in these function structures. The first structure shows a concept where a human manually breaks the composite with a tool. The second structure shows an automated breaking concept where a robot tends a press. Following the flows from left to right reveals the low-level manufacturing task sequence. In this example, two uniquely different concepts are generated by the same component database and set of rules.

Chapter 4: Knowledge-Based System Implementation

The mechanism for function structure generation must handle a wealth of design information, particularly knowledge about previous design solutions that are understood by domain experts. This knowledge exists in three main categories:

- (1) *processing function types* – general functions from common manufacturing tasks and specific functions from prior nuclear procedures (plus associated flows),
- (2) *component types* – the system components previously used for manufacturing and nuclear processes, including human and robotic operators, and
- (3) *precedence of operations* – from procedural experience.

For our application, much general manufacturing knowledge and specific nuclear processing information was obtained from operations at LANL [LANL, 2008a][LANL, 2008b][LANL, 2012]. Knowledge also comes from the author's experience with systems and gathering knowledge from engineers and scientists who have worked on projects such as MOX-type fuel pellet pressing, americium recovery and oxide conversion, plutonium electrorefining, and others related to casting and machining.

Gathered function and flow information drives the content of the other knowledge categories (Figure 4.1). The common language and associated functional/task knowledge can be used to write the knowledge-base rules that assemble function structures and derive the precedence information built into the rules. Function and flow information indicates possible embodiments, so the common language flows can also be utilized to structure and define database objects. Similar to storing empirical knowledge about previous products for reuse in new designs [Kurtoglu, 2005b], we store the previous task/processing information mentioned above to develop new manufacturing system

designs. A summary of how knowledge needs for concept generation relate to our selected artificial intelligence computational tool (i.e. KBS) is shown in Figure 4.1.

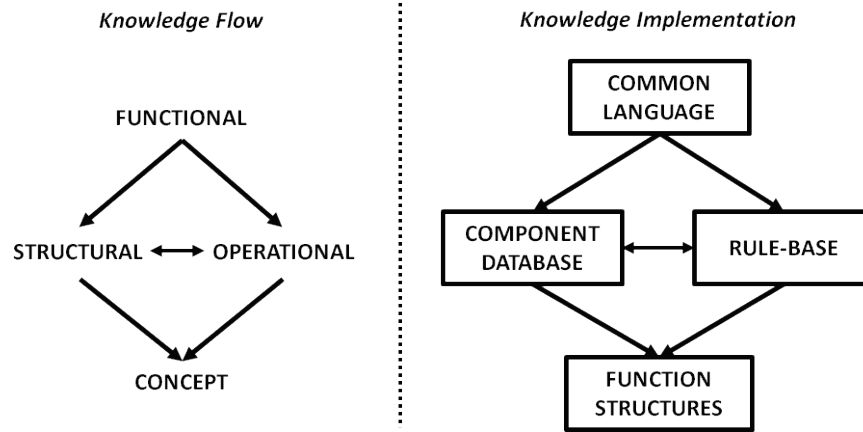


Figure 4.1: Flow and implementation of design knowledge.

FBS modeling driven by an underlying KBS that encompasses all this knowledge can provide the needed functionality for generating concepts. Expert or knowledge-based systems are well suited for ill-structured problems where there is no efficient algorithmic solution [Giarratano, 2000], as in the case of conceptual design problems. The manner in which a rule-based KBS develops a set of designs is very similar to that of grammars. In both cases, the general approach to handling function structure development is rule-based transformations on a graph structure.

4.1 KNOWLEDGE REPRESENTATION AND PROGRAMMING LANGUAGE

The success of our KBS implementation relies on the selected knowledge representation. Three common representations are rules, frames, and semantic nets [Pham, 1988][Giarratano, 2005]. Rules consist of two parts: the “if” part or antecedent

(premise or condition) and the “then” part or consequent (conclusion or action) [Negnevitsky, 2004]. This structure makes it easier to capture knowledge that is expressed in natural language such as experiential knowledge from experts.

Logic programming languages such as PROLOG utilize backward chaining to start at a goal or conclusion and work backward to determine if the conditions exists to prove the conclusion [Russell, 1995]. Production systems such as CLIPS and OPS-5 execute forward chaining to reason from existing facts/conditions to reach a conclusion that is typically a recommended action [Russell, 1995][Giarratano, 2005]. For rule-based formulations, the functionality of a production system is most relevant to our design problem.

In regard to other knowledge representations, frames and semantic nets provide declarative relational knowledge and organize the knowledge in a domain. A frame is a data structure most similar to the idea of classes in object-oriented programming – a structure for storing common generic information about an object or group to create specific instances of them. Typically, an object will have a number of attributes with corresponding values. A semantic net consists of nodes and arcs where links between nodes represent particular relationships between objects. The nodes can represent physical objects, concepts, situations, or even attributes or values [Giarratano, 2005]. The relationships represented by arcs can mean a specific object “is an instance of” another generic class, a generic object “is a kind of” another generic object, or an object “has-a” particular attribute [Giarratano, 2005].

Many current implementations adapt both rule-based and frame-based techniques for a more powerful representation. In particular, object-oriented classes provide a means of the data storage abilities of frames combined with the relational information of semantic nets. Using rules and classes, we can engineer our knowledge-base to handle the

complexities of our design problem. Multi-domain knowledge and function and flow types for modeling can be stored in frames/class objects which provides attribute inheritance through “is a kind of” relationships. The production rules can account for the dependencies of function and embodiment and store operation precedence information and other relationships.

Artificial intelligence programming languages can be used for symbolic manipulation of knowledge represented in objects and rules [Russell, 1995][Giarratano, 2005]. They can be used to build expert systems but are not expert system shells themselves [Giarrantano, 2005]. An expert system shell provides the programming language and tools to develop a user-defined knowledge-base that can be run with a built-in inference engine. To implement our KBS, we utilize the CLIPS expert system shell [Riley, 2013]. CLIPS can represent knowledge using rules and/or objects, which is desired in our KBS implementation. CLIPS is written in C and can be integrated with other languages such as Java and C++.

4.2 COMMON LANGUAGE IMPLEMENTATION

The common functional language has been implemented in CLIPS. Each function term is a CLIPS class called a *defclass*. Object attributes are defined by *slots* in the *defclass*. There are two types of *slots*: a *slot* stores a single value while a *multislot* can store multiple values. The most abstract class is the FUNCTION class that stores representation information in a number of slots: *verb*, *noun*, *input*, *output*, and assigned *component* (Figure 4.2). The slots also store criteria values related to functional and task requirements/constraints. Process, feasibility, and geometric constraints must all be considered [Rocha, 1997]. Some task requirements, such as *reach*, *payload*, and

temperature, take on corresponding quantitative values. Other values are rated on a scale from 1 to 3 based on predefined categories (see Table 3.1). Uninitialized values or values not relevant to a function receive a “nil” or “9999”.

<pre>(defclass FUNCTION (is-a USER) (slot verb (default nil)) (slot noun (default nil)) (multislot input (default nil)) (multislot output (default nil)) (slot component (default none)) ;Requirements (slot reach (default 9999)) (slot payload (default 9999)) (slot dexterity (default 9999)) (slot grasps (default 9999)) (slot accuracy (default 9999)) (slot repeat (default 9999)) (slot velocity (default 9999)) (slot motion (default 9999)) (slot vision (default 9999)) (slot force (default 9999)) (slot temp (default 9999)) (role concrete))</pre>	<pre>(defclass CHANNEL (is-a FUNCTION) (pattern-match non-reactive) (role abstract)) (defclass MOVE (is-a CHANNEL) (pattern-match reactive) (slot verb (type SYMBOL) (allowed-values Move)) (slot level (type SYMBOL) (allowed-values Terminal)) (slot from) (slot to) (slot mode (default d)) (multislot origin) (multislot materials) (multislot score) (role concrete))</pre>	<pre>([Move-1] of MOVE (verb Move) (noun Door) (input Press) (output Press) (component Man) (level Terminal) (reach 9999) (payload 3) (dexterity 2) (grasps 1) ...) ([Move-2] of MOVE (verb Move) (noun Med-mat) (input Med-mat) (output Med-mat) (component SIAS) (level Terminal) ... (from Med-cont-2) (to Med-crucible-1) (mode c) (origin 50 0 20) (materials ...))</pre>
--	--	---

Figure 4.2: The FUNCTION class template, the CHANNEL and MOVE sub-classes, and instances of the MOVE class called *Move-1* and *Move-2*.

A hierarchy is established where each function is a sub-class of one of the eight Functional Basis categories. For example, in Figure 3.2, all non-terminal and terminal functions in the *Channel* category are sub-classes of the CLIPS CHANNEL class using the *is-a* slot. Such designations allow sub-classes to inherit attribute slots from their super-classes. As function classes become more specific, additional slots are added which supplement the inherited slots. In particular, *Move* functions have slots to store information for workcell layout. CLIPS can also constrain the slot value types (ex:

symbol, integer, etc) and the values themselves for validation checking. A function class instance has specific values assigned to its slots. Default values are used when slot values are not explicitly stated in an instance initialization. As seen in the examples in this chapter, function (and flow) instances are represented in CLIPS by a string in parentheses containing a unique instance name, its class, and slot values.

Slots also exist for the layout design phase which primarily uses *Move* functions. This includes what flow the movement is *from*, the flow moved *to*, and the operator's *origin* for a *Move*. The *materials* slot stores the initial locations of nuclear materials for a *Move*. We will describe how these slots are utilized in layout optimization in Chapter 7.

4.3 COMPONENT DATABASE

The component database contains information about the flows that perform manufacturing tasks and the flows processed during these tasks. The functions in the common language help dictate the included flows. In the database, these components are generic “place-holders” that represent a family of component types. They have an approximate or preliminary form that would be replaced by detailed structures later in the design process. The generic nature of components is an advantage when performing system layout since component geometry is typically approximated in many techniques.

At the conceptual level, a main concern is the feasibility of an operator type working with component/apparatus types. The level of component abstraction can still provide useful information for this assessment. The general component structure can indicate the typical additional processing tasks that must be assigned to operators to manipulate the component. This includes whether lids or doors are present and the general size of a component or its features.



Figure 4.3: A muffle furnace [www.coleparmer.com], an induction furnace, and a hot plate [www.enasco.com].

Qualitative task compatibility criteria can be derived from such form information. In our formulation, only the essential component properties and form remain, and although the function requirements are coarse, they are specific enough to eliminate particular component types. This is because different component families/types can have discernibly different requirements, features, and capabilities. For example, in glovebox work, furnaces fall into different categories (Figure 4.3). One category is a muffle or bench-top furnace which resembles a miniature oven. This oven is a relatively compact and box-shaped with a pull down door and can sit on the glovebox floor. Induction furnaces have a larger footprint, are cylindrical in shape, and protrude out the glovebox floor. They have a “cell head” that is placed over the furnace cell and tightened when in use. Also, there are typically rods running through the cell head for electrodes and stirrers and their feedthroughs must be tightened and loosened.

As seen in Figure 4.3, the muffle furnace has a simpler form than the induction furnace. Access to the muffle furnace only requires manipulation from the front to open

and insert material and there is good clearance for these tasks. The door trajectory is not very difficult, although grabbing the handle requires a fingertip grasp. The induction furnace requires a large number of attainable hand orientations to loosen the bolts holding down the cell head and the fasteners for the cell head feedthroughs. The space is much more cramped for manipulation and fingertip grasps are needed for some tasks while two hands are needed to lift the cell head. Thus, based on Table 3.1, the *dexterity*, *grasps*, and *motion* slots for operating the muffle furnace could be 2, 1, and 1, respectively, and 3, 3, and 2, respectively, for the induction furnace. A human or manipulator arm with a basic gripper could feasibly open and use the muffle furnace, while only a human could probably open the induction furnace in its current configuration. We could even consider a hot plate for heating and how its operations are much simpler than both furnace types.

Furthermore, the temperature requirement and the materials processed and produced (I/O) can also filter furnace types. Certain muffle furnaces may not be able to reach the temperatures that some induction furnaces can. Also, a muffle furnace can convert an oxalate to an oxide and an induction furnace can electrorefine a material, but neither should be used for the other's task. Even with basic forms and component types, a level of operator-component compatibility and function-component compatibility can be established at this conceptual stage.

The templates for component/flow types are stored as general CLIPS classes named using flow terms from the common language. Other work has also utilized the Functional Basis for a component database/repository [Kurtoglu, 2009]. Flows have slots with criteria values for constraint checking against function requirement slots as well as slots for component physical properties/parameters and state information. The *dexterity*, *grasps*, and *motion* slots for apparatuses are typically reserved for defining the requirements for working with that apparatus (Figure 4.4). The other requirements slots

(*accuracy*, *repeatability*, etc) define the abilities provided by the apparatus. If a flow has a sub-target (*Lid*, *Door*, or *Lock*) in its *subtargets* slot, these are defined separately as SUBTARGET instances (bottom of Figure 4.5). Their *dexterity*, *grasps*, and *motion* slots correspond to the task capabilities needed for opening and closing the sub-target of the apparatus or container. Any number of attributes can be added to classes as slots. Whether or not a particular slot is utilized depends on the desired model complexity and/or implemented functionality in the knowledge-base.

<pre>(defclass FLOW (is-a USER) (pattern-match non-reactive) (slot id) (slot title) (multislot subtargets) (slot loc_iter (default 1) (type INTEGER)) (slot theta (default 0)) (multislot location (default (create\$ 0 0 0))) (slot status (default m)) (role abstract))</pre>	<pre>(defclass OPERATOR (is-a COMPONENT) (pattern-match non-reactive) (slot reach (default 9999)) (slot payload (default 9999)) (slot dexterity (default 9999)) (slot grasps (default 9999)) (slot accuracy (default 9999)) (slot repeat (default 9999)) (slot velocity (default 9999)) (slot motion (default 9999)) (slot vision (default 9999)) (slot force (default 9999)) (slot temp (default 9999)) (slot inside) (role abstract))</pre>	<pre>(defclass APPARATUS (is-a COMPONENT) (pattern-match reactive) (slot opened (default False)) (slot load_mass (default 0)) (multislot contains) (slot size (default 9999)) (multislot dim) (multislot circles) (multislot access_loc (default (create\$ 0 0 0))) (slot dexterity (default 9999)) (slot grasps (default 9999)) (slot motion (default 9999)) (slot accuracy (default 9999)) (slot repeat (default 9999)) (slot velocity (default 9999)) (slot vision (default 9999)) (slot force (default 9999)) (slot temp (default 9999)) (slot payload (default 9999)) (role concrete))</pre>
<pre>(defclass COMPONENT (is-a FLOW) (pattern-match non-reactive) (multislot classes (type SYMBOL)) (multislot used_by) (multislot using) (role abstract))</pre>		

Figure 4.4: The high-level FLOW class and some sub-classes.

The component database in CLIPS contains generic instances of the component classes/types with unique component names and specific slot value definitions. For instance, the *SIA5* robot in Figure 4.5 is a member of the ROBOT flow category/class of the common language. It has specific function categories it is allowed to embody (*classes*

slot) and is also rated for the approximate task abilities it can provide for a function (*reach*, *grasps*, etc). During embodiment, a component (including operators) can fulfill the function in question when its requirement slot values meet or exceed the desired requirement slot values of the function. Branching of feasible designs occurs when multiple components exist with the necessary criteria values to perform a function.

<pre> (defclass HUMAN (is-a OPERATOR) (pattern-match reactive) (multislot hand_locs) (role concrete)) (defclass ROBOT (is-a OPERATOR) (pattern-match reactive) (slot dof) (multislot mounts) (multislot dim) (slot alpha (default 0)) (multislot circles) (multislot access_loc (default (create\$ 0 0 0))) (multislot endeffectors) (multislot eef_locs) (role concrete)) </pre>	<pre> (Man of HUMAN (id 1)(title Man)(inside Glovebox-1) (classes BRANCH CHANNEL CONNECT CONTROL SUPPORT ACTIVATE DEACTIVATE DETECT STABILIZE) (reach 70)(payload 10)(dexterity 3)(grasps 3)(accuracy 2)(repeat 2) (velocity 2)(motion 3)(vision 3)(force 2)(temp 150)) (SIA5 of ROBOT (id 1)(title SIA5) (classes BRANCH CHANNEL CONNECT SIGNAL SUPPORT STABILIZE) (dof 7)(endeffectors Parallel Hand) (reach 80)(payload 5)(dexterity 3)(grasps 2)(accuracy 3)(repeat 3) (velocity 3)(motion 2)(vision 9999)(force 9999)(temp 9999)) (Press-1 of APPARATUS (id 1)(title Press-1)(classes BREAK DEFORM) (subtargets Door)(force 3)(repeat 3)(velocity 1)(temp 300)(size 3)(dim 15 10 20)) (Furnace-1 of APPARATUS (id 1)(title Furnace-1) (subtargets Door)(classes HEAT) (accuracy 2)(velocity 9999)(vision 9999)(temp 1200)(size 2)(dim 15 10 20)) (Glovebox-1 of GLOVEBOX (id 1)(title Glovebox-1) (ports 18 0 20 64 0 20 100 0 20 146 0 20)(dim 164 152 122)) (Med-cont-2 of CONTAINER (id 5)(title Med-cont-2) (classes STORE SUPPLY)(contains Med-mat)(mass 2)(grasps 2)(size 2)) (Press-1-Door of SUBTARGET (id 6)(title Press-1-Door) (mass 0)(grasps 2)(dexterity 2)(motion 2)(size 1)) </pre>
---	--

Figure 4.5: Two OPERATOR sub-classes and some flow instances.

The handled flows (see Figure 4.6) also help determine the requirements to assign to functions and dictate the functional decomposition. In particular, the mass and general size of a transported material defines the *payload* and *grasps* slot values/requirements for the *Move* function. Also, the ability to *Store* a material in a container can depend on the comparison of *size* or *dim* (dimension) slots, if available. If a non-solid flow such as a

powder or collection of loose materials is handled or transferred, different functions are required (*Dispense* rather than *Transport*).

<pre>(defclass MATERIAL (is-a FLOW) (pattern-match non-reactive) (slot mass (default 9999)) (slot grasps (default 9999)) (slot size (default 9999)) (slot dose (default 0)) (slot inside) (role abstract))</pre>	<pre>(defclass COLLECTION (is-a FLOW) (pattern-match non-reactive) (slot mass (default 9999)) (slot grasps (default 9999)) (slot size (default 9999)) (slot inside) (multislot contains) (role abstract))</pre>	<pre>(defclass ITEM (is-a FLOW) (pattern-match non-reactive) (slot mass (default 9999)) (slot grasps (default 9999)) (slot size (default 9999)) (multislot dim) (multislot circles) (role abstract))</pre>
<pre>(defclass MOBJECT (is-a MATERIAL) (pattern-match reactive) (multislot dim) (multislot circles) (role concrete))</pre>	<pre>(defclass COMPOSITE (is-a COLLECTION) (pattern-match reactive) (multislot dim) (multislot circles) (role concrete))</pre>	<pre>(defclass PART (is-a ITEM) (pattern-match reactive) (slot inside) (role concrete))</pre>

(Small-mat-1 of **MOBJECT** (id 1)(title Small-mat-1)(mass 1)(grasps 1)(size 1))
(Large-mat of **MOBJECT** (id 4)(title Large-mat)(mass 8)(grasps 3)(size 2))
(Compos-1 of **COMPOSITE** (id 1)(title Compos-1)(contains Small-mat-1 Med-mat)(mass 3)(grasps 2)(size 2))
(Part-1 of **PART** (id 1)(title Part-1)(mass 1)(grasps 1)(size 1))
(Assembly-1 of **ASSEMBLY** (id 4)(title Assembly-1)(contains Part-1 Part-2)(mass 9999)(grasps 9999)(size 9999))

Figure 4.6: Some classes for processed flows and particular instances.

By grouping components based on function, structures with different behaviors are grouped together. When embodiment occurs, a component must first be capable of providing that function in a general sense, independent of behavior. Then functional requirements are checked for specific function-component compatibility. Thus, when linking a function to a structure, the requirements can indirectly filter out some behaviors. The behavior of the selected structure may also influence some of the additional functions needed in the function structure and differences in the behaviors of structures for a given function may lead to different performance ratings during design evaluation.

For example, one component may be safer than another for human use based on the physical effect it utilizes to execute a function. In this way, behavior acts indirectly in the transformation from function to structure.

4.4 RULES

The knowledge-base rules contain and utilize extensive knowledge to generate embodied function structures. The rules determine when and how non-terminal functions should decompose and also maintain the precedence of operations. This heuristic information is gained from published work, experience, and discussions with engineers involved in these processes. The rules also utilize the knowledge stored in function and flow classes/instances to make embodiment and operational decisions. The effectiveness of these rules depends greatly on the selection of function and component slot types. The requirements slots help the rules make informed decisions about function-component and function-operator compatibility. Other slots help the rules make processing decisions based on component and material states. These include whether an apparatus or container is opened or closed and whether one flow is stored in or storing another flow.

4.4.1 General Rule Formulation

Based on such knowledge, as the function structure develops the knowledge-base rules determine whether function decomposition, embodiment, or addition should occur for a queried function in the function structure. The *decomposition rules* determine when a higher-level function should be decomposed into sub-functions. For example, the decomposition of *Break* functions in Figure 3.5 and Figure 3.6 or the decomposition of a *Load* (apparatus) function into *Detect* (material to load into apparatus), *Move* (material),

and *Store* (material in apparatus). The *embodiment rules* find database components and operators that are compatible with a given function. The *add rules* determine if additional functions must be inserted into the function structure based on the properties of the selected embodiment. For instance, if a *Press* must *Break* a material, a *Load Press* function is inserted directly in front of the *Break* function to place the material inside the *Press*. A *Close* function may also be added between *Load* and *Break* if a press shroud is present and should be closed to keep materials from ejecting from the *Press* when broken. Once this higher-level processing information is added, further function decomposition and embodiment can occur.

A general overview of how the rules function together is seen in Figure 4.7. Typically, the overall process function is non-terminal and not embodied. Once this function is embodied, it usually requires additional functions based on the state of its component or the flows it processes. These added flows are generally non-terminal and not embodied. They are then decomposed and embodied to generate a set of finished functions (i.e. terminal and embodied). Work on these additional functions changes the states of the flow used by the original embodied non-terminal function which now allow this function to be decomposed. During decomposition, some embodied terminal functions are directly produced while some require an extra embodiment step. At this point, the rule application process is complete. The task plan order of finished functions is generally given as left to right in Figure 4.7.

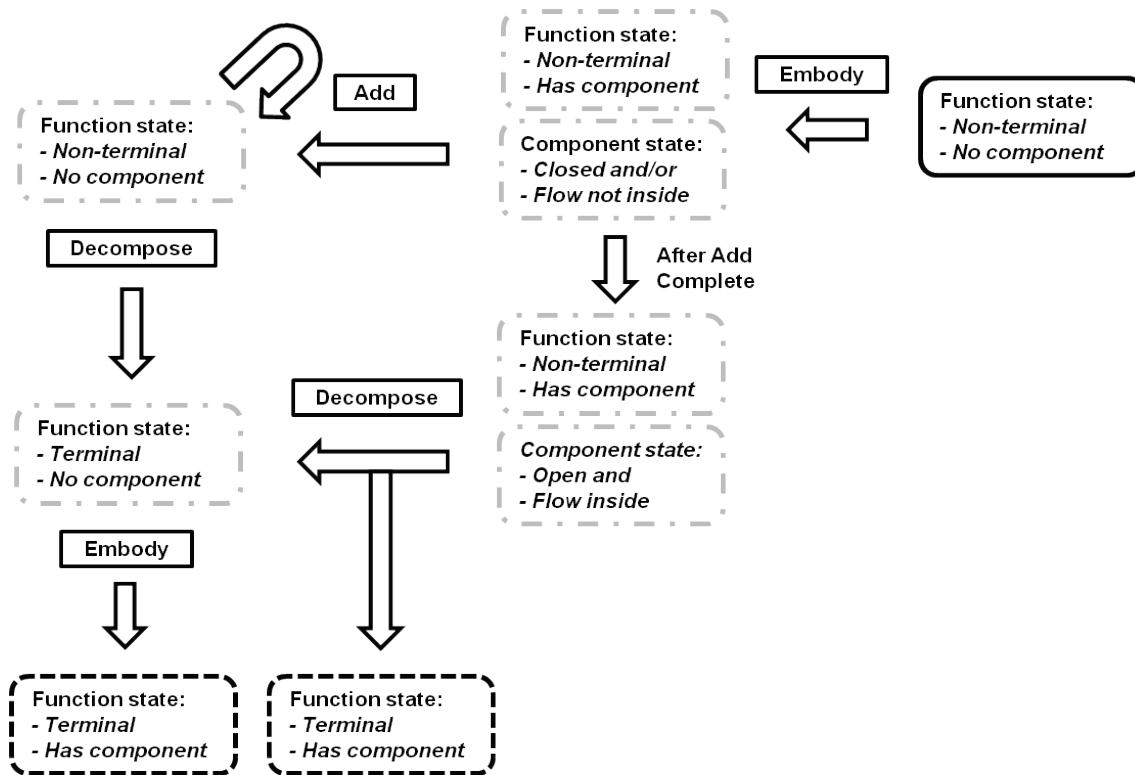


Figure 4.7: The general sequencing of rule application.

The rules also determine the input and output flows for newly created functions. To perform design automation, consistency in the input/output combinations for particular function types is maintained throughout the rules. This was developed from knowledge of task types and experience with test cases. Whenever a function is decomposed, the original inputs and outputs are present in the decomposition. If a non-terminal function is embodied, then the embodiment flow is typically added to the input and output to allow added functions (*Load*, *Open*, etc) to be connected.

Rules in CLIPS are written in an “If-Then” form. The left-hand-side (LHS) of a rule specifies what values (properties or states) an instance/object (function or flow) must possess in its slots for the rule to apply. The searched instances include any functions

added to the CLIPS environment and all the component instances from the database (which are always present). To find applicable instances, the built-in inference engine scans the slot values of rules and instances for matching patterns (i.e. the existence of certain slots and slot values). For potential matches, the rules also contain conditional tests to query instance states, such as if a component has been assigned to a function, if a material is present in an apparatus, if an apparatus is opened/closed, if a particular requirement is met, etc. The rule conditional elements also control the sequence in which rules can be applied in order to maintain the precedence of process operations.

The rule's right-hand-side (RHS) describes what should be executed if one or more objects match the pattern and satisfy the conditional tests on the LHS. The execution of the RHS is also called *rule firing*. Forward-chaining is used to execute sequences of LHS pattern matching and RHS firing to update function and/or component slots and add functions. From a design perspective, a rule applies to a function in the function structure and the rule firing results in a design/function structure update. Then this pattern match and execution sequence continues to run for each updated function structure until the design concept is complete.

4.4.2 Embodiment Rules

To demonstrate the interactions of functions, components, and rules in the CLIPS environment, we use an example embodiment rule (Figure 4.8). The LHS (before the “=>”) of this rule searches the CLIPS environment for two objects (*?ob1* and *?ob2*). The slots after the “<-” define what slot values these objects must possess. When slot values are defined, such as “Terminal” for *level* and “none” for *component* in *?ob1* (1), only function instances possessing those values (or pattern) in their respective slots are

possible matches. The other slot values beginning with “?” are variables that store the object’s value for that specific slot instead of requiring a particular value. These variables can be used elsewhere in the rule (LHS or RHS) for comparisons and/or value assignments. Thus, the presented rule searches for a flow instance (*?ob2*) of the OPERATOR class (2) that can embody a terminal function instance (*?ob1*).

```
(defrule EMBODY-operator [6] [1]
  ?ob1 <- (object (is-a ?R_class) (noun ?noun)(level Terminal)(component none)
    (payload ?R_payload)(grasps ?R_grasps)(dexterity ?R_dexterity)(motion ?R_motion))
  [2] [5]
  ?ob2 <- (object (is-a OPERATOR) (title ?C_title)(location $?C_location)
    (payload ?C_payload)(grasps ?C_grasps)(dexterity ?C_dexterity)(motion ?C_motion))

  ;Does not apply to Detect and Store functions
  (test (and (not (eq ?R_class DETECT)) [3]
    (not (eq ?R_class STORE)) ))

  ;Check requirements
  (test (or (<= ?R_payload ?C_payload) [4]
    (eq ?R_payload 9999)) )
  (test (or (<= ?R_grasps ?C_grasps)
    (eq ?R_grasps 9999)) )
  (test (or (<= ?R_dexterity ?C_dexterity)
    (eq ?R_dexterity 9999)) )
  (test (or (<= ?R_motion ?C_motion)
    (eq ?R_motion 9999)) )

=>
  ;Embody function with component
  (send ?ob1 put-component ?C_title) [5]

  ;Update Move for configuration
  (if (eq ?R_class MOVE) [6]
    then

    (send ?ob1 put-mode c) [7]

  ) ;End if
)
```

Figure 4.8: An example operator embodiment rule.

The LHS *test* statements (3 & 4) check particular relationships among slot values of matched instances. The first test checks that the function class of *?ob1* is not DETECT or STORE – there are separate embodiment rules for these functions. The next set of tests

compares the function task requirement values (starting with “?R_”) to the task abilities (starting with “?C_”) of the operator *?ob2* (4). The operator’s abilities must meet or exceed those required by the function, and if the value is not defined (i.e. equals 9999) for a particular requirement, that requirement is ignored. The rule does not apply for a given function-component (*?ob1*-*?ob2*) combination if at least one instance slot does not match a defined slot value or a test fails.

If a matched operator in the database meets the functional/task requirements, the rule’s RHS fires to assign the operator’s *title* (*?C_title*) to the function’s *component* slot using the CLIPS function *send* (5). Otherwise, the rule does not allow the operator to embody/perform that function. If the function class is MOVE (6), the RHS also updates the *mode* slot to *c* (7) to signal that this *Move* function is ready for the configuration/layout stage (to be described in Chapter 7). Both (5) and (7) demonstrate how instance slots are set to update functions. Similar slot value changes are made for flows to change their physical states (*mass*, *opened/closed*, *inside*, etc).

The inference engine tracks all feasible combinations of functions and/or components that apply for a rule. How the CLIPS environment finds matches using a rule such as that in Figure 4.8 is shown in Figure 4.9. A function instance called *Move-1* has been added to the environment (1). Also shown is a sample of component database instances present in the environment (2). On the right side (3) are abbreviated left-hand-sides of rules called *EMBODY-operator* (from Figure 4.8), *EMBODY-detect*, and *dec-STABILIZE*.

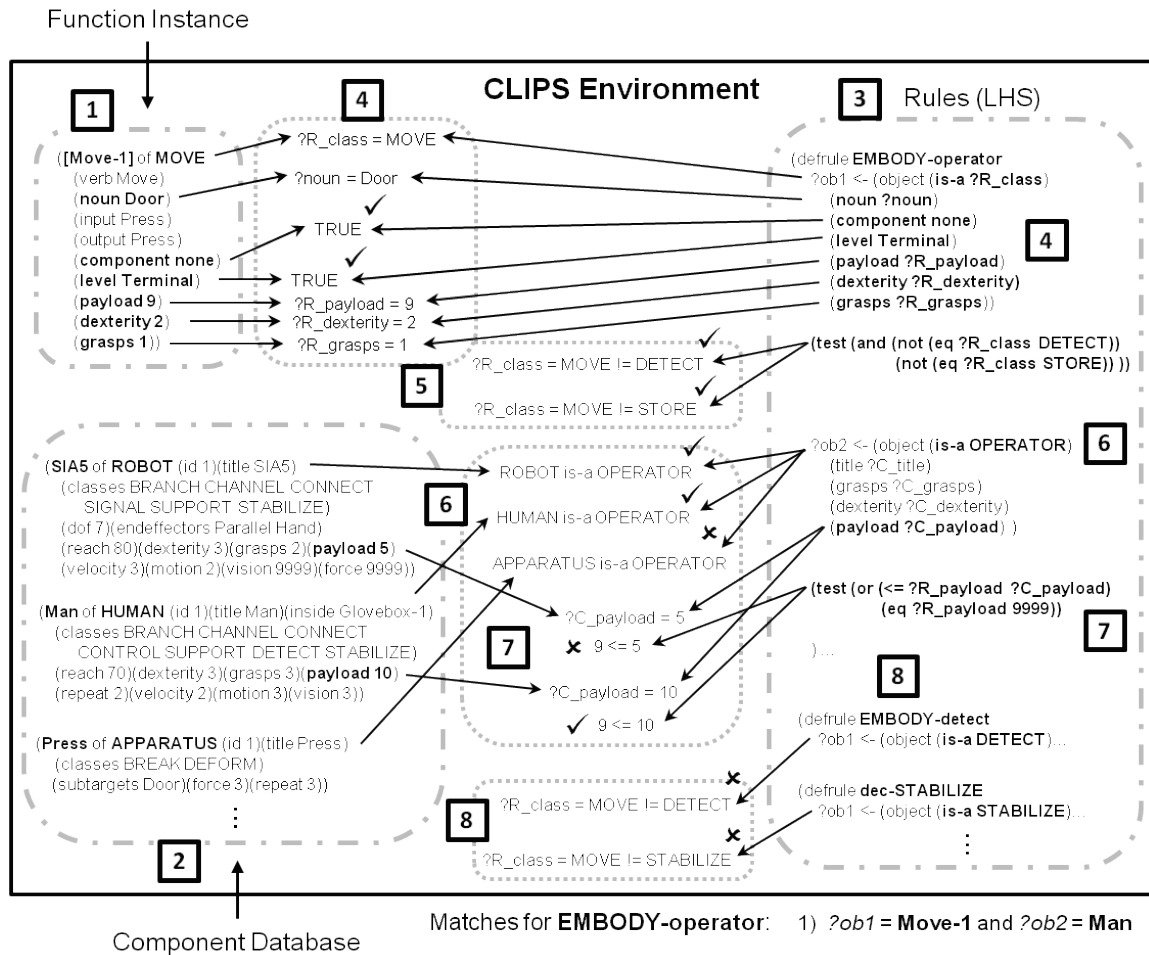


Figure 4.9: An example of the LHS of rules interacting with instances in the CLIPS environment.

The *EMBODY-operator* rule begins by comparing its *?ob1* slots to those of *Move-1* (4). Defined slots are compared to see if they have the same value and the rule variables (with “?”s) are assigned values from the corresponding slots of *Move-1*. Then the *?ob1* class is checked to make sure it is not DETECT or STORE (5) using the class name that was stored earlier in the *?R_class* variable. Everything checks out, so *Move-1* is a valid function instance for *?ob1*.

Then the inference engine uses the rule's *?ob2* pattern (slots and slot values) to search the component database for an OPERATOR instance (6). The ROBOT and HUMAN classes are sub-classes of the OPERATOR class (as defined in the common language), so the *SAI5* and the *Man* are valid OPERATOR instances. However, the APPARATUS class of the *Press* is not a sub-class of OPERATOR, so it cannot match the *?ob2* pattern. Next, the payload requirements are tested (7). When an instance is matched to *?ob2*, its payload slot value is assigned to the variable *?C_payload*. Using the *test* statement, this operator payload ability is compared to the payload needed by the function which is stored in *?R_payload* during (4). The *SIA5* fails to meet the payload requirement, but the *Man* does not. Therefore, in this abbreviated case, the only pattern match found for the LHS of the *EMBODY-operator* rule is *Move-1* for *?ob1* and *Man* for *?ob2*. Other rules are filtered out by their slot values. For instance, the other two rules do not apply since only a function of the MOVE class is present and not DETECT or STABILIZE (8).

The results of rule firing are seen in Figure 4.10. The RHS changes the *component* and *mode* slot values, moving the function to a new state. No other instances have slot values changed. In terms of the function structure, the *Move* function now has component assigned to it (*Man*).

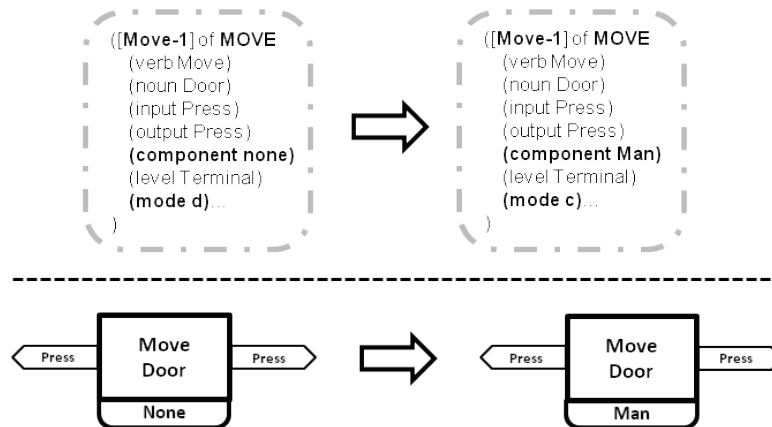


Figure 4.10: The changes caused by rule firing in CLIPS code (top) and the function block representation (bottom).

The pseudo code for the *Break* embodiment rule is seen in Figure 4.11. Important checks and operations are performed for each searched object. If too many flows are in the function output, this rule would not apply, but instead the *Break* would be decomposed into multiple *Break* functions using a decomposition rule. When searching the database for *Object 2*, the rule first looks for a component that can fulfill the function in a broad sense using the *classes* slot and then confirms more specific compatibility by checking functional requirements. When defining new functions on the RHS, many slot values are defined using the variables from the objects on the LHS. This includes the functional representation slots (*verb*, *noun*, *input*, *output*, and *component*) and the requirement slots (*payload*, *grasps*, etc). Thus, the rules contain information about how the input and output flow values should be defined to ensure that the proper flows are passed on to subsequent functions. In this way, relevant information is retained and propagated as the function structure develops.


```

(defrule EMBODY-break

*Search for combination:

  Object 1 (Break function)
    - Function class BREAK
    - No component assigned to component slot
    - Number of output flows less than or equal to 2

  Object 2 (Object to embody Break)
    - Flow class APPARATUS or TOOL
    - BREAK is a member of classes slot
    - Apparatus or tool-operator combination meets applicable requirements

  Object 3 (Object being broken)
    - Same name as noun of Break
    - Object is inside something
    - Object will fit inside Object 2

*End search

=>

*If found:

  If Object 2 is a tool, then
  Create Stabilize function instance
    - Set noun, input, and output equal to value of Object 1's noun slot
  End if

  Create new Break function
    - Add Object 2's title to input and output
    - Set component slot to Object 2's title

  Delete Object 1

*End if

)

```

Figure 4.11: The *Break* embodiment rule.

The decomposition can be seen graphically in Figure 4.12. In this way, the original *Break* function is replaced by a *Stabilize* function and a new embodied *Break* function with updated inputs and outputs. Thus, decomposition and embodiment occur in this step. The order in which the new functions are created determines the precedence among those functions – the order in which functions are created is their order when inserted back into the function structure for the function they replace. Requirements such as *grasps* and *payload* can be updated in this step for propagation, if desired.

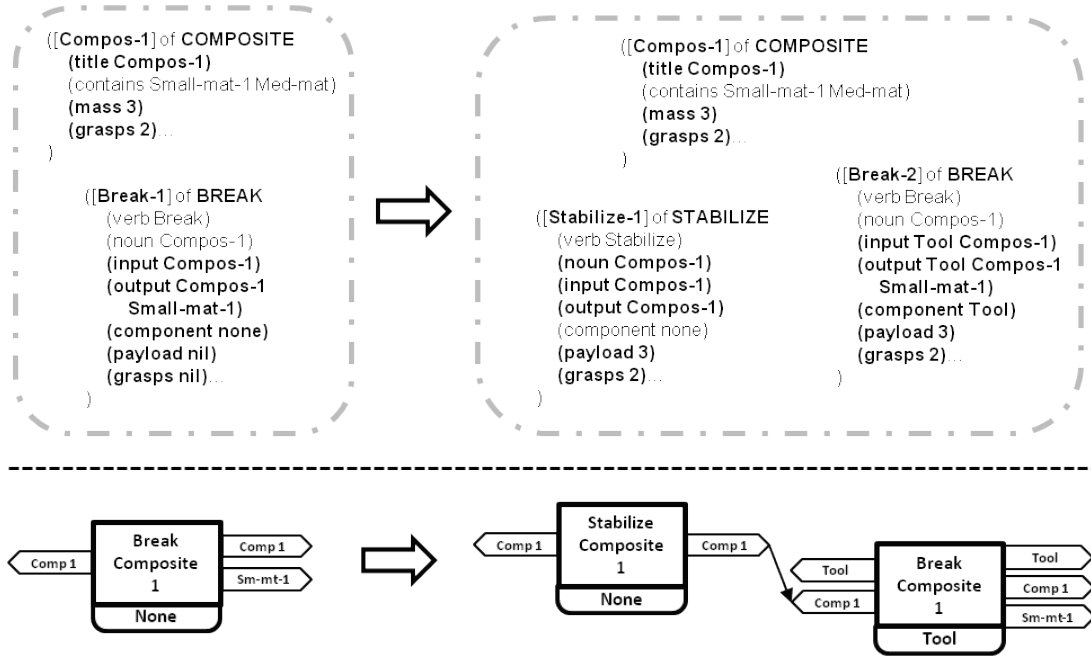


Figure 4.12: The initial and final states after the *EMBODY-break* rule is applied.

4.4.3 Add Rules

The add rules are typically triggered based on the physical states of one or more instances and implement common material handling functions. These include rules to add *Open*, *Close*, *Load*, *Unload*, and *Store* functions. The slots frequently utilized by such rules are *inside*, *contains*, *subtargets*, and *opened*. The *inside* slot stores the flow that a particular flow is inside, and the *contains* multislot stores what flows a particular flow is holding/storing. The *subtargets* multislot stores whether or not a flow has a *Door*, *Lid*, and/or *Lock* that must be opened or closed. The opened/closed state of a flow's *subtargets* is stored by the *opened* slot.

(defrule add-OPEN-con

*Search for combination:

Object 1 (*Connect* or *Convert* function)

- Function class is a sub-class of CONNECT or CONVERT
- Has a component assigned to *component* slot

Object 2 (Object 1's embodiment)

- Object's *title* is equal to value in Object 1's *component* slot
- Object is of class APPARATUS or CONTAINER, has *subtargets*, and *opened* slot is "False" (i.e. it is closed)

Object 3 (Object being processed)

- Object's *title* is equal to value in Object 1's *noun* slot
- Object is not currently inside Object 2
- Object's *title* is not a member of Object 2's *contains* slot

*End search

=>

*If found:

Create *Open* function instance

- Set *noun*, *input*, and *output* to value of Object 2's *title* slot

Set Object 2's *opened* slot to "True"

Duplicate Object 1

- Give new function instance a unique name

Delete Object 1

*End if

)

Figure 4.13: An add rule that inserts an *Open* function.

The add rule in Figure 4.13 can insert an *Open* function when a component used to *Heat* is closed and the flow it acts on is not yet loaded inside of it. The rule applies to all functions that are sub-classes of CONNECT or CONVERT and encounter this state. In this rule, the LHS object conditions help maintain the proper operational sequence – that is, they prevent other rules from firing until this rule fires and changes function and/or component states so that subsequent rules can be activated. The *Heat* function must already be embodied for this rule to apply, and *Object 3* cannot already be loaded

into *Object 2*. Once this rule adds an *Open*, then a *Load* function can be called, and finally a *Close* function. All of these functions must first be performed in sequence prior to running the apparatus. In this way, the rules check the current state of components to identify the next step needed in the overall process and then find rules that apply to that specific sub-process. The addition of *Open* followed by the duplication of *Heat* then deletion of the original *Heat* function preserves the processing order.

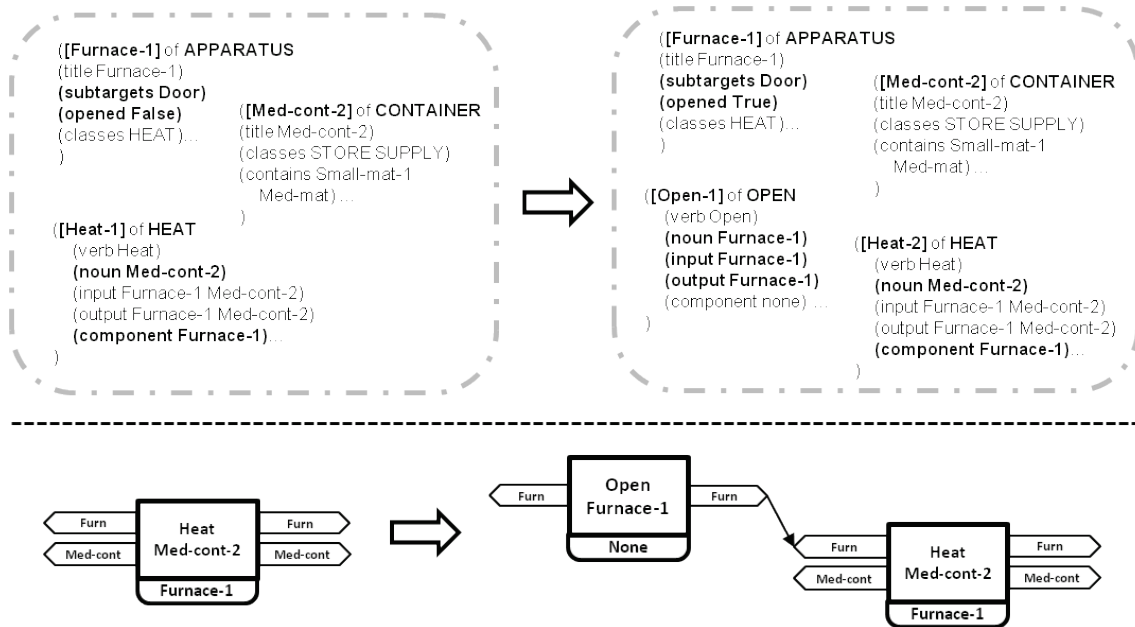


Figure 4.14: The results of firing the *add-OPEN-con* rule for a *Heat* function.

Other *add-OPEN* rules work similarly. Essentially, the LHS searches for an apparatus performing a function with one or more closed sub-targets. Some rules have an additional check to see if the flow to be used is inside a different apparatus or container that is closed. In this case, that flow is opened too. Open rules also apply to *Load* and *Unload* functions when their *nouns* are closed. For instance, this could be applied when

loading materials in separate containers into a single, currently closed container (the *noun* of *Load*). There are also *add-LOAD* and *add-UNLOAD* rules. These rules check for similar instances and instance states as the *add-OPEN* rules, except they typically require that the object/flow to be loaded or unloaded is already open or does not require opening (i.e. its *subtargets* slot is empty).

4.4.4 Decomposition Rules

When certain conditions are met for a function and the flows it processes, functional decomposition rules execute. The decomposition rules essentially contain modular groups of functions that replace another function. They are designed to be useable in a variety of situations and are the building blocks that help make up the whole function structure. These rules are utilized to piece together a function structure rather than “hard-code” large parts of it. When this “hard-coding” is avoided, more creative designs are generated. Creativity in designs comes from two main sources. Different embodiment options may lead to different added/supporting functions: *Open/Close*, *Load/Unload (Transport)*, *Stabilize*, etc. In turn, more creativity is achieved by the operator selections available for these material handling tasks. Thus, the decomposition rules contain small chunks of “hard-coded” functions, but decisions about if these chunks should be applied, what order they are applied, and how they are applied depend on the combination of components and operators selected from the database.

The more basic decompositions are for material handling tasks, while other decompositions apply to embodied, non-terminal functions. Decomposition rules typically apply when no more functions need to be added, as the flows processed by a function are in a certain state: opened or closed, contain the proper flows, etc. There are

also decomposition rules that break a single function with multiple inputs and/or outputs into multiple functions of the same type each with fewer inputs and/or outputs. *Store* and *Supply* functions are frequently incorporated into decomposition. The *Store* function is used to indicate a flow's location or storage inside another component. The *Supply* function is called when a stored flow is made active again (i.e. is needed by another function).

```
(defrule dec-TRANSPORT
```

```
  *Search for combination:
```

```
    Object 1 (Transport function)
```

- Function class TRANSPORT
- Has no component assigned to *component* slot

```
    Object 2 (Object transported)
```

- Object's *title* is equal to value in Object 1's *noun* slot

```
  *End search
```

```
  =>
```

```
  *If found:
```

```
    Create Detect function instance
```

- Set *noun*, *input*, and *output* slots to the same values as Object 1

```
    If Object 2 is a container, then
```

```
      Set the Move payload slot to the sum of the container mass and the mass within the container
```

```
    Else
```

```
      Set the payload slot to the mass of Object 2
```

```
    End if
```

```
    Create Move function instance
```

- Set *noun*, *input*, and *output* slots to the same values as Object 1
- Set the *from* and *to* slots according to Object 1's *from* and *to* slots
- Set the *payload*, *grasps*, *dexterity*, and *motion* slots based on the corresponding *from* and *to* component slot values

```
    Delete Object 1
```

```
  *End if
```

```
)
```

Figure 4.15: A decomposition rule for *Transport*.

Figure 4.15 contains a decomposition rule for a non-embodied *Transport* function. This decomposition rule executes after other rules that add *Load* and then decompose *Load* (which produces a *Transport*). The *dec-TRANSPORT* rule is a final decomposition rule since it produces only terminal functions (Figure 4.16). Such decomposition rules commonly update slot values related to requirements and physical states. In this example, the *Move* function has its *payload*, *grasps*, *dexterity*, and *motion* slots updated because the next step for *Move* is embodiment using these requirements. Thus, at some point during decomposition depending on the state of information and when it will be needed, slot values are updated in preparation for embodiment. Sometimes requirements are propagated through a series of functions using their slots.

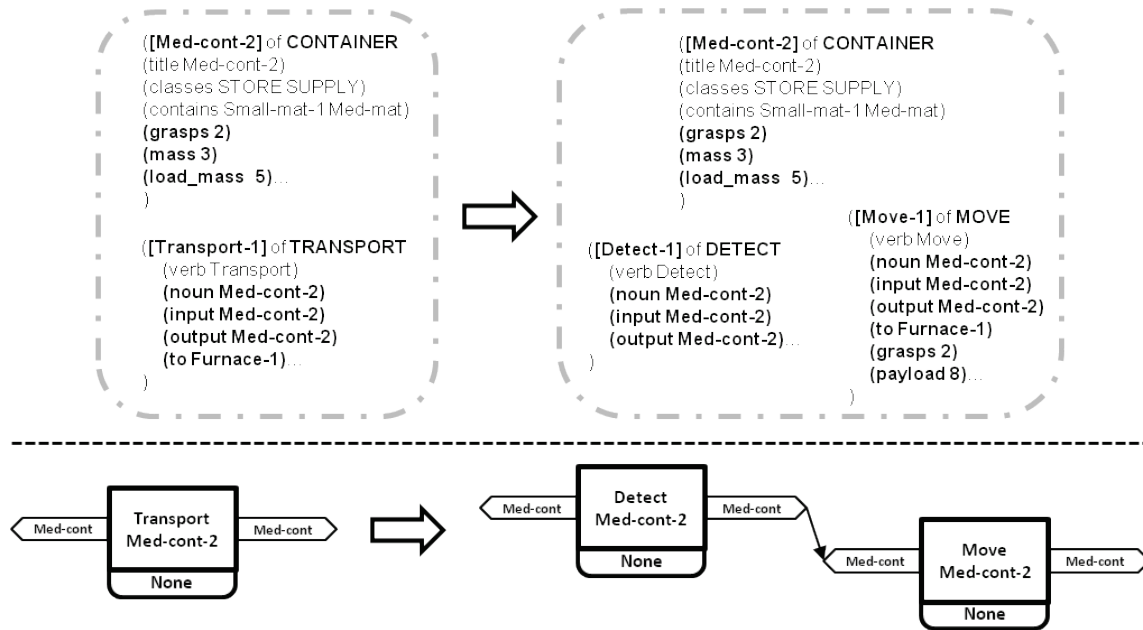


Figure 4.16: The results of firing the *dec-TRANSPORT* rule.

Other decomposition rules have stricter LHS conditions and more complex decompositions. Decomposition rules may incorporate many conditional statements because different material handling tasks exist depending on what apparatus, containers, and/or processed flows are involved. This is seen in the decomposition of *Unload* in Figure 4.17. In this case, *Object 2* and *Object 3* are the two flows used in the *Unload* function. Different RHS decompositions are generated depending on whether *Object 3* is a solid or loose object and *Object 2* is an apparatus or container. Three of these possibilities are shown in Figure 4.18. Also, the *contains* slot is updated for *Object 2* when handling solid objects but not loose objects. The *contains* slot (and others) for the latter case is updated during decomposition of *Dispense*. This decomposition produces terminal functions, thus the requirements are updated right before embodiment. Although the same decomposition rule can apply to many function and component combinations, the rule may produce several different decompositions and changes in flow states based on what is processed. The RHS essentially contains all enumerations for the possible sub-tasks and final states of a higher-level operation given the function and initial flow states.

When an embodied non-terminal function is decomposed, additional operations are performed. The original function is typically decomposed into an embodied terminal function (ex: *Create*, *Separate*, or *Join*) plus other supporting functions (see Step 6 of Figure 3.5). The rule also manipulates the flows involved in the physical process performed by the component: taking material inputs to a *Heat* function and creating a composite containing those materials, creating an assembly containing two parts, removing two materials from a composite, etc. All of these changes of state are recorded as updates to *inside* and *contains* slots.

(defrule dec-UNLOAD-to-load

*Search for combination:

Object 1 (*Unload* function)

- Function class UNLOAD
- Has no component assigned to *component* slot
- The number of outputs equals 2

Object 2 (Object containing Object 3)

- Object's *title* is equal to value in Object 1's *noun* slot
- The *opened* slot is "True" or the object has no sub-targets to open

Object 3 (Object removed from Object 2)

- Object's *title* is one of Object 1's outputs (along with Object 2's *title*)

*End search

=>

*If found:

If Object 2's class is a CONTAINER or APPARATUS and Object 3 is a solid object, then

- Set Object 2's *load_mass* slot to its current *load_mass* value minus Object 3's *mass*
- Remove Object 3's *title* from Object 2's *contains* slot

End if

If Object 3 is a solid object or CONTAINER, then

Create *Supply* function instance

- Set *input* and *output* slots to the same values as Object 1
- Set *noun* equal to Object 3's *title* and *component* equal to Object 2's *title*

Else (for non-solid objects)

Create *Dispense* function instance

- Set slot values similarly as *Supply* but *component* to "none"

End if

If Object 3's class is CONTAINER and Object 2's class is APPARATUS, then

Create *Transport* function instance

- Set *noun*, *input*, and *output* slots to the same value as Object 3's *title*
- Set *from* equal to Object 2's *title* and *component* equal to "none"

End if

Delete Object 1

*End if

)

Figure 4.17: A decomposition rule for *Unload*.

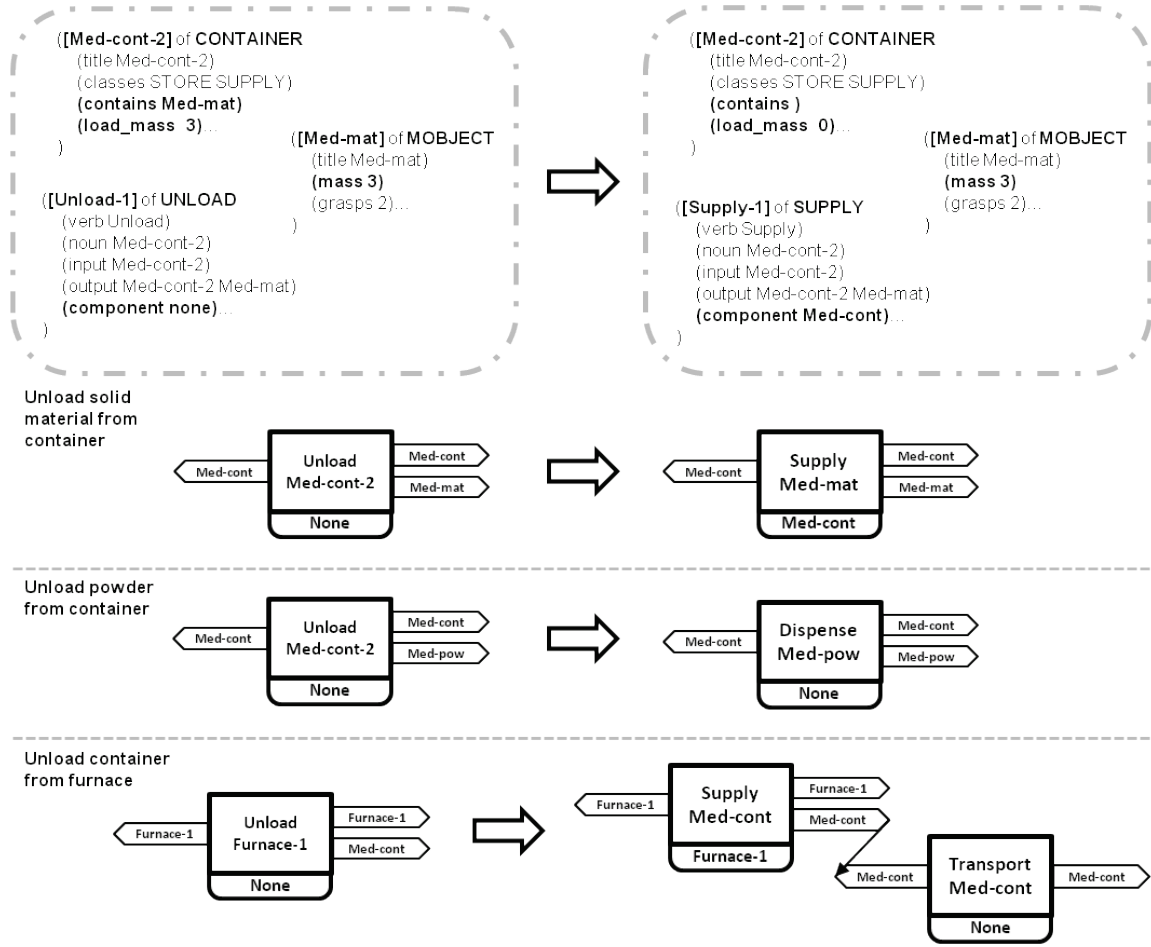


Figure 4.18: Three different decompositions for *Unload* depending on the involved flows.

4.5 KNOWLEDGE-BASE ADDITIONS

Rule contents will need to be updated if additional tasks and component types need to be incorporated into the knowledge-base. Continued work with the rule-base will determine what refinements are necessary. For example, after extensive trials, we may determine a lower level decomposition should be added to and performed in a higher level decomposition or that some conditions can be removed from the LHS of certain

rules. Thus, the possibility of discovering new needs or making changes to the rule-base always exists.

When new knowledge is needed, the KBS framework allows straight-forward implementation since the structure is flexible and designed for extensions. For example, to add functionality for *Mix* functions, a few additions are made. *Mix* flows/components must be added to the component database. These include powders, a mixer, and a mixer lock SUBTARGET instance (if needed). The appropriate slot values should also be populated. An embody rule is added to assign components to the *Mix* function. All flows processed by *Mix* must be in a single container, so the *add-LOAD-container* function is updated to pattern match on *Mix* functions since this rule previously only dealt with *Heat* functions. Since the operations surrounding *Mix* are similar to *Heat*, code is updated in other *Heat* rules and reused to create the new rules *add-LOAD-heat/mix*, *dec-HEAT/MIX-new*, and *dec-HEAT/MIX-container*. Thus, no new rules were created besides embodiment since similarities in processing meant previous code could be reused. The main challenge in knowledge-base additions lies in rule changes. The conditional tests must be modified and tested to ensure that they support the new types of processing introduced while maintaining operational precedence and previously supported processing.

4.6 CONTROL PROGRAM

CLIPS is embedded into C++ code that manages design development and storage. There are two C++ classes to handle functions and designs (function structures). The MFUNCTION class stores available data about CLIPS function instances. The main object attribute is a string variable named *DES* for storing the entire CLIPS instance

description which includes *name*, CLIPS *class*, and all other slots names and values. Other support exists for getting and setting values, printing values to the screen, etc.

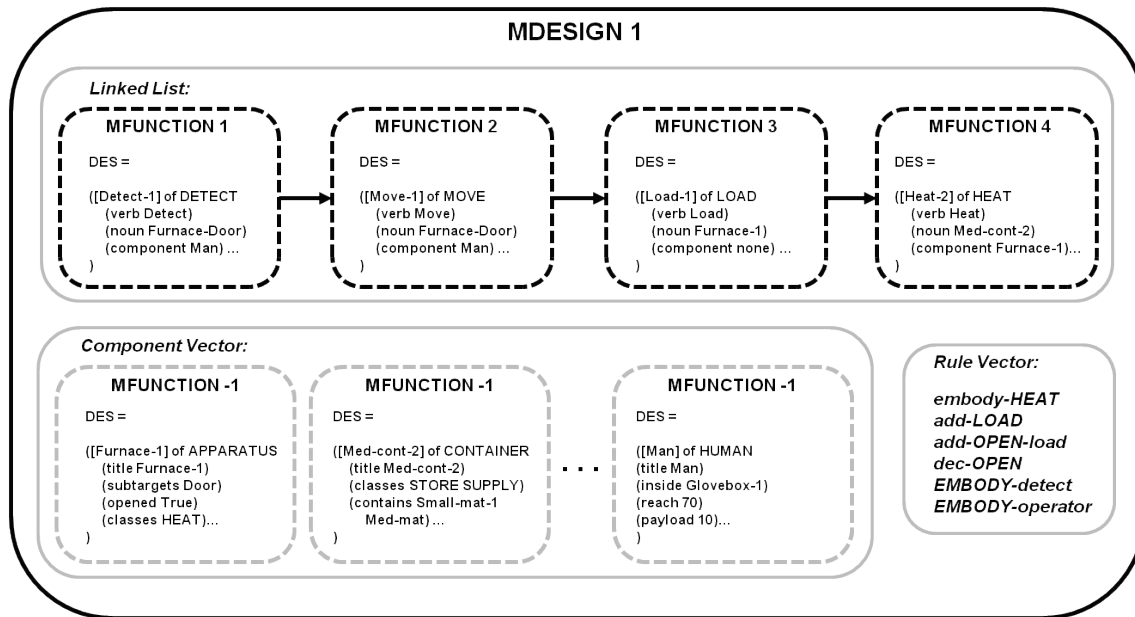


Figure 4.19: An example of a MDESIGN object.

The MDESIGN class stores objects from the MFUNCTION class in a C++ *linked list* unique to the MDESIGN object (Figure 4.19). Thus, each MDESIGN object represents a design concept where the linked list gives the current function structure for that design. The MDESIGN object also stores the rules fired by CLIPS to generate its function structure and has supporting functions for manipulating the linked list. Since multiple design concepts can exist at one time in the C++ program, the MDESIGN object also stores the states of the database components it currently uses. The MFUNCTION object is utilized to store component instances/data, saving the component's representation from CLIPS as its *DES* attribute. Thus, when a new design is being

worked on, the component database resets in CLIPS then the MDESIGN object sends the component string representations to CLIPS to re-write all components used by the design. All component-related MFUNCTION objects are stored in a C++ *vector*.

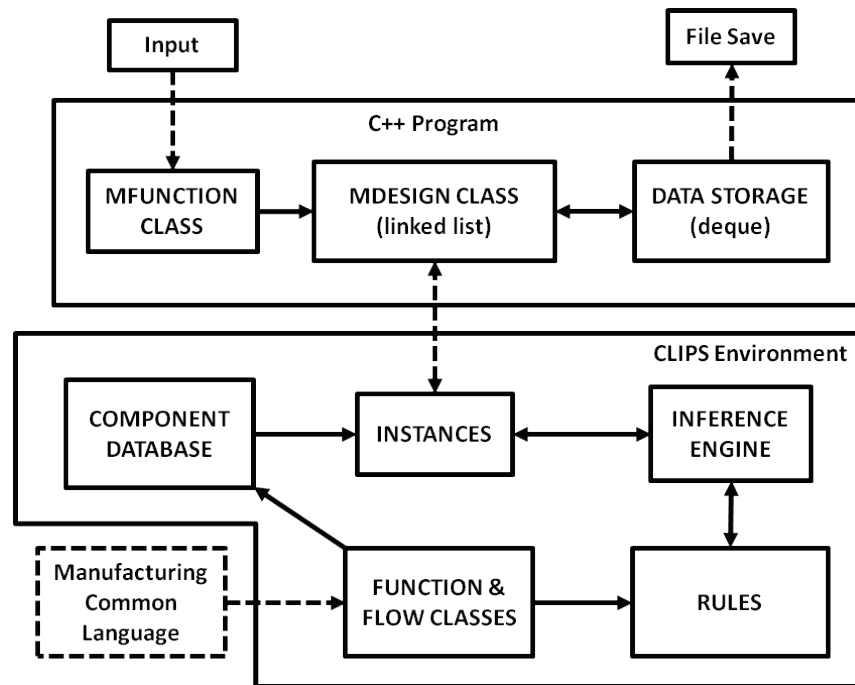


Figure 4.20: The CLIPS and C++ implementations for our KBS.

The C++ program works directly with CLIPS during function structure generation (Figure 4.20). The process starts by adding one or more MFUNCTIONs to the linked list of a MDESIGN. One at a time, beginning from the front of the linked list, each non-terminal and/or non-embodied function (the active function) is passed to CLIPS and created as a CLIPS instance/object (Figure 4.21). The inference engine then searches for rules whose conditions are satisfied by the slot values of one or more instances (function and/or component/flow). CLIPS then sends the number of applicable rules back to the

C++ program. The C++ code then makes that number of copies of the current MDESIGN. Then, for each new MDESIGN, the corresponding rule is fired in CLIPS, modifying slot values and/or adding new functions. The function instances left over are then passed back to the control program and inserted in the proper linked list location. For example, if an embodiment rule activates three times for a function for three different components, three new MDESIGNS are created, and each has a different component embodying that function. The components used in each design are also sent to and stored in the corresponding MDESIGN. After each MDESIGN is updated, it is pushed onto the back of a C++ *deque*¹ and the old, copied design is deleted. Then, the MDESIGN on the top of the *deque* is “popped off” and the next non-terminal and/or non-embodied function is passed to CLIPS. The rule activation and execution cycle is repeated and the *deque* is properly modified.

A MDESIGN “popped off” the *deque* is complete when the function structure only contains terminal functions all with assigned components. Information from the popped MDESIGN is instead saved to two types of files. The full solution file lists the information from the *DES* variable of all MFUNCTIONs in the linked list from beginning to end. That is, the CLIPS instance description is written for each function. A low-level manufacturing task sequence is also constructed from the linked list (function structure) which gives the order of functions to execute, what component or operator performs that function, and what flows/objects are handled or manipulated when the function is performed. This task sequence is added to a separate file by only writing the *component*, *verb*, and *noun* for each function instance description to generate a phrase for

¹ In C++, a *deque* is a storage structure representing a double-ended queue.

each task. Also included in this file is the ordered list of the rules applied to generate the solution.

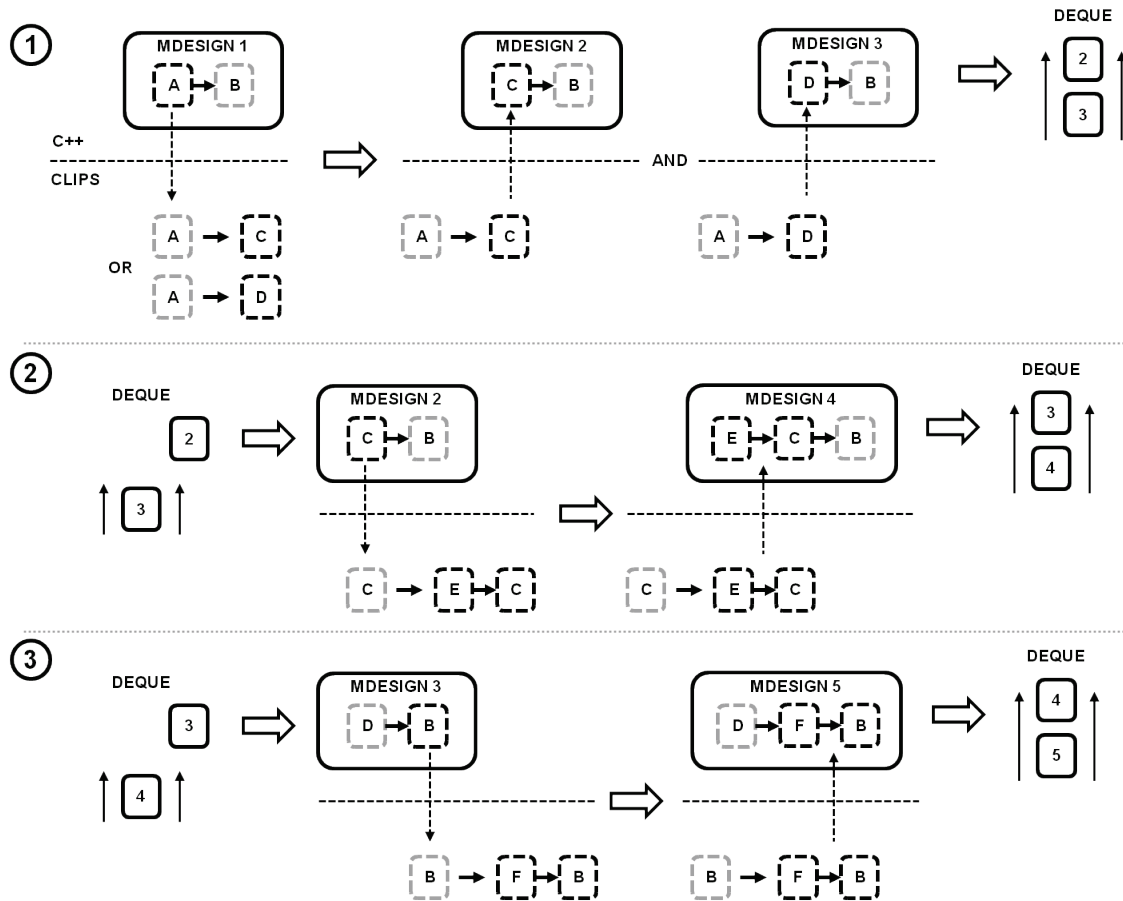


Figure 4.21: An example of passing functions between C++ and CLIPS.

The manner in which CLIPS operates and interfaces with C++ dictates how the combined CLIPS and C++ code must function. This particularly affects the need to send functions to CLIPS one at a time. The rules pattern match on all instances present in the CLIPS environment, so a full design, and especially multiple designs, cannot be present if we want to focus on a specific part of the function structure for a particular design.

Furthermore, deciding on and controlling the order in which multiple matched rules would fire is complicated. Thus, only one function is sent and the rules are designed so that multiple rules are applicable simultaneously only if they are function embodiment rules matched for multiple compatible components. Thus, only one *add* or *decomposition* rule is applicable at one time. Otherwise, if two different types of rules fire, for instance, the two new MDESIGNS created would only have one rule applied instead of both.

4.7 SUMMARY

This chapter presented all the components needed to build our concept generation technique. The automation of FBS modeling dictated many implementation needs, particularly the development of the common manufacturing language and a KBS to build designs. Multi-domain knowledge is included throughout all KBS modules. The overall goal is to use knowledge about common manufacturing tasks and component types to generate creative or untried combinations, particularly those that utilize robotics. This requires gathering information from experience and experts to define the functional, component, and operational knowledge that must be stored for the KBS to effectively generate concepts.

Chapter 5: Concept Generation Examples

We will now present concepts generated using the KBS described in Chapter 4. The current KBS can generate embodied function structures and low-level task plans for a variety of general manufacturing tasks. To demonstrate its functionality, we use the KBS to produce system design concepts for manufacturing processes similar to those at LANL.

5.1 KBS INPUT

The KBS receives inputs from the C++ control program to begin. These inputs are strings that CLIPS can interpret as *defclass* instances. The function strings are defined according to the CLIPS function templates. These are specific functions and are the main/required tasks of the overall process and should be from the second non-terminal column in the common language. The non-terminal material handling functions are filled in by the program as needed by the input functions. *Store* functions can also be defined in the input to constrain where particular flows are moved to or store a product in a final location. Otherwise, the program automatically chooses what containers to store flows in or use for an apparatus.

The input order of functions determines which are manipulated first. The input functions must have the *noun*, *input*, and *output* slots defined. All the slot values must be component *titles* from the database. The *noun* slot contains a single noun (database component) and varies among functions. For BRANCH functions, the noun is the input, and for CONNECT and CONVERT functions, the noun is the output (which is derived from the input). The *component* slot is only defined if a *Store* function is used. The *Store* function takes two inputs, the second being the object being stored (the noun). The

correct manner to define all function *noun*, *input*, and *output* slots will not be described here but would be communicated to the user.

The requirement slots (ex: *grasps*, *temp*, *force*, etc) of input functions can be filled in as desired to restrict embodiment selections. The rules automatically handle slot assignments for material handling and terminal functions. Also, the *inside* and *contains* slots typically do not need to be populated in the database – the rules manage these slots during generation. This overall input formalism coincides with our concept generation goals. We must adhere to a specific process, so particular main tasks must be defined (the input), but there is flexibility in the material handling operations that supplement them (added by the KBS).

There are a few details the KBS user must know when using the program besides the types of inputs the program accepts. The KBS user must know what possible components/flows exist in the database and have knowledge of what requirements should be defined to constrain the components found by the embodiment rules. Also, to avoid incompatibilities, rules will not embody with a component if a requirement is defined in the function but that slot value is not defined for the component. Depending on the process tasks, enough components must be defined in the database to embody all the expected functions, otherwise complete designs cannot be generated. Additionally, if different requirements exist for the same function applied to different flows, these must be defined separately. This is due to KBS rules that decompose BRANCH and CONNECT functions/classes with more than two processed flows into multiple functions with at most two processed flows.

The KBS contains the common supporting functions for manufacturing processes and the ability to add specific functions/sub-processes. Thus, a list of high-level processes (ex: “Electrorefine Pu” and “Produce Am Oxide”) recognized by the KBS for a specific

domain (ex: nuclear) could be built into the knowledge-base. In this case, the KBS input in a single high-level process whose first decomposition would result in the types of input functions described above. However, we present our examples at the functional level to demonstrate generality and applicability to general manufacturing.

5.2 RESULTS

The following examples demonstrate the information complexity managed by the KBS when generating solutions and the diversity of the resulting concept types.

5.2.1 Single *Break* Example

We begin with a general example to demonstrate the types of concepts the KBS generates and how different modules function together. As seen at the top of Figure 5.1, the input is a *Break* function where a composite (*Compos-1*) is broken into two material objects (*Small-mat-1* and *Med-mat*). The *force*, *accuracy*, and *motion* requirements are also defined with values corresponding to a more rough breaking process without a high degree of precision. Important database components include the human operator (*Man*), two robots (*SLA5* and *SLA10*), a breaking tool (*Chisel*), a breaking press (*Press-1*), and various containers.

Input:
(Break of BREAK (input Compos-1)(output Med-mat Small-mat-1)(noun Compos-1)(force 2)(accuracy 2)(motion 2))

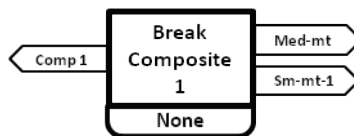


Figure 5.1: The string input for the *Break* function and a block representation.

CONCEPT 1:

Med-cont-2 Store Compos-1
Man Move Man
Man Grasp Compos-1
 Compos-1 Supply Small-mat-1
Man Move Chisel
Chisel Separate Small-mat-1
Man Release Med-mat
 Med-cont-2 Store Med-mat
 Med-cont-2 Store Small-mat-1

Rules 1:

ADD-store-initial
 EMBODY-store-initial
 EMBODY-break
 EMBODY-stabilize
 dec-STABILIZE
 dec-BREAK-single
 EMBODY-move

CONCEPT 2:

Med-cont-2 Store Compos-1
 Med-cont-2 Supply Compos-1
 Man Detect Compos-1
Man Move Compos-1
Fixture-2 Store Compos-1
 Fixture-2 Supply Compos-1
 Fixture-2 Support Compos-1
 Compos-1 Supply Small-mat-1
Man Move Chisel
Chisel Separate Small-mat-1
 Fixture-2 Store Med-mat
 Fixture-2 Store Small-mat-1

Rules 2:

ADD-store-initial	(cont.)
EMBODY-store-initial	dec-TRANSPORT
EMBODY-break	EMBODY-detect
EMBODY-stabilize	EMBODY-move
add-UNLOAD	dec-STABILIZE
dec-UNLOAD-to-load	dec-BREAK-single
add-LOAD-stabilize	EMBODY-move
dec-LOAD-single	

CONCEPT 3:

Med-cont-1 Store Compos-1
 Med-cont-1 Supply Med-cont-1-Lid
 Man Detect Med-cont-1-Lid
Man Move Med-cont-1-Lid
 Man Detect Fixture-1-Lock
Man Move Fixture-1-Lock
 Med-cont-1 Supply Compos-1
 Man Detect Compos-1
Man Move Compos-1
Fixture-1 Store Compos-1
Man Detect Fixture-1-Lock
Man Move Fixture-1-Lock
 Fixture-1 Supply Compos-1
 Fixture-1 Support Compos-1
 Compos-1 Supply Small-mat-1
Man Move Chisel
Chisel Separate Small-mat-1
 Fixture-1 Store Med-mat
 Fixture-1 Store Small-mat-1

Rules 3:

ADD-store-initial	(cont.)
EMBODY-store-initial	add-LOAD-stabilize
EMBODY-break	dec-LOAD-single
EMBODY-stabilize	dec-TRANSPORT
add-OPEN-branch/stab	EMBODY-detect
dec-OPEN	EMBODY-move
EMBODY-detect	add-CLOSE
EMBODY-move	dec-CLOSE
dec-OPEN	EMBODY-detect
EMBODY-detect	EMBODY-move
EMBODY-move	dec-STABILIZE
add-UNLOAD	dec-BREAK-single
dec-UNLOAD-to-load	EMBODY-move

Figure 5.2: Some concepts for breaking a composite utilizing only human labor.

Concept 1 in Figure 5.2 represents the most basic manual case. *Compos-1* sits in an open container (*Med-cont-2*) and the *Man* grasps the composite with one hand then moves the *Chisel* to separate the *Small-mat-1* from the *Med-mat*. The precise nature of the breaking movement is unknown. The rule application to build this solution is also

straight forward. The KBS first finds a container to store *Compos-1* inside. Then an embodiment is found for the *Break* function. In this case, a tool (*Chisel*) is selected, leaving the need to stabilize *Compos-1*. The *Man* is selected to stabilize *Compos-1* for the break. Now the *Break* function can be decomposed and the separated materials are stored in the same container as the original composite.

Concept 2 shows the difference in the task plan if a fixture is utilized to stabilize *Compos-1* instead of the *Man* grasping it. In this case, *Compos-1* is moved from its container (*Med-cont-2*) to a fixture (*Fixture-2*) that it can fit inside. Then the breaking proceeds as in *Concept 1*.

The third concept shows the additional tasks needed if lids and locks are present. In this example, the lid must be removed from the container storing *Compos-1* (*Med-cont-1*) before *Compos-1* can be moved to a fixture. Also, the fixture (*Fixture-1*) has a lock to hold down loaded objects. This lock is opened, *Compos-1* is loaded, the lock is fastened to secure *Compos-1*, and then the breaking can begin.

We will use *Concept 3* as an example to show how C++ and CLIPS pass information back-and-forth to generate concepts. The initial component database is loaded into CLIPS. The *Break* function (user input) is defined as a CLIPS string and stored as the *DES* attribute for a C++ MFUNCTION. This MFUNCTION is then loaded into an C++ MDESIGN object and the program begins. Only one MFUNCTION exists in the design. It is set as the *active function* in the linked list and sends the string from its *DES* slot to CLIPS. This string, the code for a CLIPS *Break* function, is converted to a CLIPS instance. The rules scan the *Break* instance slots and the slots of component database instances. Based on the precedence information built into the rules, it finds that the *Break* function uses a flow that should be stored inside a container but it is not (i.e. the *inside* slot of *Compos-1* is “nil”). Thus, the first rule to fire creates a *Store* function

for *Compos-1* with its *component* slot set to “none”. Then the original *Break* function is duplicated then deleted. Since *Store* was created before the new *Break* function, it will be inserted into the MDESIGN linked list before *Break*, saving the function order. (It is important to remember that copies of the original MDESIGN are created for each activated rule (reference Figure 4.19), but we only focus on the function interactions in this example.) When the *Store* function and new *Break* function are sent back to C++, their CLIPS strings are saved to the *DES* attribute for two new and separate MFUNCTIONS. They are inserted in the same location of the linked-list as the original *Break* function. The strings for all the components used by the design (in this case only *Compos-1*) are also sent back to the MDESIGN to save their state (slot values) for that particular MDESIGN.

The next active function (front-most non-embodied and/or non-terminal function) in the MDESIGN is *Store* since it is not embodied. When it is sent back to CLIPS, the database also resets and the flows used by the MDESIGN are sent back and overwrite the existing flows so that all flow states are up-to-date for that particular MDESIGN. The inference engine finds a rule that applies to a *Store* function, a flow not inside anything, and an empty container. The compatible component selected from the database is *Med-cont-1*. During this rule firing, the *inside* slot of *Compos-1* is set to *Med-cont-1* and the *contains* slot of *Med-cont-1* is set to *Compos-1*. This embodied *Store* function is sent back to the linked-list and inserted in its original location before *Break*.

The *Store* function is terminal and embodied, so the next function sent back to CLIPS is *Break* once again. The current information is insufficient to determine processing needs (i.e. allow add or decomposition rules), so the *Break* function is embodied. The embodiment rule takes the requirements from the *Break* function and scans the database for components with those capabilities and BREAK in their *classes*

slot. A compatible operator and tool combination (*Human* and *Chisel*) is found and the tool's *title* is set as the value in the *Break* function's *component* slot. Additionally, since a tool was selected, a *Stabilize Compos-1* function is created. The original *Break* function is duplicated again and then deleted, so the *Stabilize* function is added to the linked list before the *Break* function but both are behind the *Store* function.

The *Stabilize* is now the active function in the list and is sent to CLIPS. An embody function finds a container (*Fixture-1*) with STABILIZE in its *classes* slot that can fit *Compos-1* and sends the embodied *Stabilize* function back to C++. The *Stabilize* function is still the active function in the design since it is non-terminal. When it is passed to CLIPS, the presence of an embodied function allows other rules to activate. *Compos-1* needs to be moved to *Fixture-1*, but it is inside *Med-cont-1* which is closed. So an add *Open* function is called which not only makes an *Open Med-cont-1* function but an *Open Fixture-1* function after it since *Fixture-1* has a lock and is currently closed. These *Open* functions are placed before *Stabilize* in the linked list. Then one at a time, each *Open* is passed to CLIPS and decomposed into *Detect* and *Move* functions. These latter two functions are sent back to CLIPS individually to find an embodiment from the database. In the process, the *opened* slots of *Med-cont-1* and *Fixture-1* are set to "True".

Once *Med-cont-1* and *Fixture-1* are open, an *Unload* function is added then decomposed to remove *Compos-1* from the *contains* slot of *Med-cont-1*. This allows a *Load Fixture-1* function to be added then decomposed, updating the *inside* slot of *Compos-1* and the *contains* slot of *Fixture-1*. The terminal functions from decomposition are subsequently embodied. Then a *Close Fixture-1* function is added to close the lock on *Fixture-1*, followed by decomposition and embodiment similar to *Load*. Essentially, whenever the rules detect no other additions/processing needs and functions are not low-

level enough for embodiment, decomposition occurs until terminal functions are reached and can be embodied.

CONCEPT 4:

Med-cont-2 Store Compos-1
 Man Detect Press-1-Door
Man Move Press-1-Door
 Med-cont-2 Supply Compos-1
 Camera-1 Detect Compos-1
SIA10 Move Compos-1
Press-1 Store Compos-1
 Man Detect Press-1-Door
Man Move Press-1-Door
 Press-1 Supply Compos-1
 Compos-1 Supply Small-mat-1
Man Activate Press-1
Press-1 Separate Small-mat-1
Man Disable Press-1
 Press-1 Store Med-mat
 Press-1 Store Small-mat-1

CONCEPT 5:

Med-cont-1 Store Compos-1
 Med-cont-1 Supply Med-cont-1-Lid
 Camera-1 Detect Med-cont-1-Lid
SIA10 Move Med-cont-1-Lid
 Camera-1 Detect Press-1-Door
SIA10 Move Press-1-Door
 Med-cont-1 Supply Compos-1
 Camera-1 Detect Compos-1
SIA10 Move Compos-1
 Press-1 Store Compos-1
 Camera-1 Detect Press-1-Door
SIA10 Move Press-1-Door
 Press-1 Supply Compos-1
 Compos-1 Supply Small-mat-1
Man Activate Press-1
Press-1 Separate Small-mat-1
Man Disable Press-1
 Press-1 Store Med-mat
 Press-1 Store Small-mat-1

Rules 4:

ADD-store-initial
 EMBODY-store-initial
 EMBODY-break
 add-OPEN-branch/stab
 dec-OPEN
 EMBODY-detect
 EMBODY-move
 add-UNLOAD
 dec-UNLOAD-to-load
 add-LOAD-branch
 dec-LOAD-single
 dec-TRANSPORT
 EMBODY-detect
 EMBODY-move
 add-CLOSE
 dec-CLOSE
 EMBODY-detect
 EMBODY-move
 dec-BREAK-single
 EMBODY-act/dis
 EMBODY-act/dis

Rules 5:

ADD-store-initial
 EMBODY-store-initial
 EMBODY-break
 add-OPEN-branch/stab
 dec-OPEN
 EMBODY-detect
 EMBODY-move
 dec-OPEN
 EMBODY-detect
 EMBODY-move
 add-UNLOAD
 dec-UNLOAD-to-load
 add-LOAD-branch
 dec-LOAD-single
 dec-TRANSPORT
 EMBODY-detect
 EMBODY-move
 add-CLOSE
 dec-CLOSE
 EMBODY-detect
 EMBODY-move
 dec-BREAK-single
 EMBODY-act/dis
 EMBODY-act/dis

Figure 5.3: Some automated concepts for breaking materials apart.

Eventually a string of embodied terminal functions are before *Stabilize* and *Break* in the linked list. These latter two functions are then sent in sequence to CLIPS for decomposition to produce the *Supply*, *Support*, *Move*, *Separate*, and *Store* functions needed to execute the actual break made by the *Man* using the *Chisel*. This concept is simply one solution branch of many created by the generation algorithm.

Figure 5.3 presents some concepts utilizing automation for the same input *Break* function. In *Concept 3*, a press (*Press-1*) performs the breaking task, while a robot (*SIA10*) loads *Compos-1* into the press and the *Man* executes all the other movement tasks. The *grasps*, *dexterity*, and *motion* requirements for *Med-cont-1*'s lid (*Med-cont-1-Lid*) and *Press-1*'s door (*Press-1-Door*) are selected such that no robots possess the task capabilities for those opening movements. Also, the mass of *Compos-1* (6 kg) excludes the *SIA5* robot (payload of 5 kg) from moving it. In this hybrid workcell, the robot (*SIA10*) does the "heavy lifting" while the human performs the finer motion tasks. Robotics use is aided by the ability to use a press, as a robot does not provide as much of a benefit when a human is using a tool.

As the requirements become less strict, we can move to other solutions in the design space. *Concept 4* in Figure 5.3 displays a fully automated design where robots perform all material handling tasks. In this case, the task requirements for opening the lid and door are more lax, which could result from changes in the component. Thus, with the right apparatus sub-targets (i.e. *Lids* and *Doors*) compatible with robotics (easy to open and in fixed places) a fully automated solution is possible. Another step toward more automation is the use of a control program to run the press (activate or disable). However, more information about the process is needed for effective implementation.

Although many requirements are not identified in the KBS input, the rules automatically attach requirements for material handling to functions based on the

processed materials. For example, grasping and payload requirements are given to *Move* functions when they are created during decomposition. Opening, closing, and transfer sequences are also added based on precedence information determined from component and material states (stored in slots). Additionally, the qualities/sub-targets of used apparatuses and containers influence operator compatibility/selection. It would be difficult for a human designer to track all of these details for every possible design.

One important point is that the task sequence for a particular design is not the only possible task sequence. There is often flexibility in how to construct the sequence. Sometimes tasks can be performed in parallel or two designers may view a function's decomposition differently. In either case, we select an order that is maintained by the rules. The goal here is not to produce *the* definitive task plan but generate one that contains all the important process considerations. In particular, our main emphasis is on the required embodiments and movements. The existence and order of functions such as *Supply*, *Store*, and *Support* is not as important as the sequencing and embodiment of main functions and material handling tasks. These latter considerations play an integral part in workcell layout.

5.2.2 Plutonium Electrorefining Example

A more complex example generates task sequences related to plutonium electrorefining (ER). In this process, an impure Pu anode heel is placed in a crucible with a salt electrolyte and seeding agent. The contents are heated in an induction furnace until molten. Then a voltage is placed across the anode and cathode to drive the transport of Pu ions through the salt from the anode to cathode, leaving impurities behind at the anode. Upon run termination and cooling, a solid mass of pure Pu, depleted anode, salt, and

crucible remains and must be separated (Figure 5.4). This process can be represented by the following abbreviated steps:

- 1) Load materials into a crucible
- 2) Heat crucible in a furnace
- 3) Unload furnace contents
- 4) Break apart solid

Based on the common language terms, Steps 2 and 4 are inputs to the KBS (Figure 5.5). The necessary material handling tasks such as Steps 1 and 3 are automatically added during the generation process.



Figure 5.4: An opened furnace cell (left) and the resulting solid from an ER run (right). The Pu is the outermost ring, followed by the crucible (white) and inner anode.

In the output, *Concept 1* of Figure 5.6 shows an option for manual/human processing only. This is how the process is currently executed in the ER glovebox at LANL. The *Man* first loads all three materials from their separate containers into a single crucible (*Med-crucible-1*). Based on the presence of a *Heat* function, the rules check to see if all inputs are not only in one container but in one that is used for heating (i.e.

contains HEAT in its *classes* slot). The *Man* then opens the furnace, loads the crucible, closes the furnace, and runs the furnace. Upon opening and unloading the furnace, the *Man* breaks apart the composite with a tool. A *Sort* could be called after the breaking to move the materials into separate containers.

Input:
 (Heat of HEAT (input Med-mat Small-mat-1 Small-mat-2)(output Compos-1)(noun Aggregate-1)(temp 800))
 (Break of BREAK (noun Compos-1)(input Compos-1)(output Med-mat Small-mat-1 Small-mat-2))

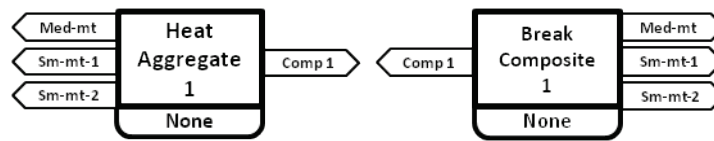


Figure 5.5: The input for the Pu ER example.

In this example, there is the possibility that the wrong type of furnace could be selected if its temperature rating meets requirements. When embodying a *Heat* function, it is harder to define requirements that filter possible embodiments (although the task requirements for embodiments are typically quite varied). This demonstrates the general principle that knowledge about the overall *process* must be incorporated at some point either by developing the KBS to receive input in the form of process descriptions or using functional inputs then sifting through solutions after concept generation.

We can take the generated design type (Figure 5.6) and examine how changes to components or the process could influence material handling and thus feasible designs. In this particular case, as the current process is run, some tasks are only suited for humans. The adjustments needed to tighten feedthroughs on the furnace cell head and the lifting of the cell head are particularly difficult and not well-suited for robotic manipulation. A change to the furnace is needed for robotic manipulation and this is most likely only

feasible through changes to the ER process. The resulting solid mass from the process is also difficult to break, requiring a lot of dexterity and vision. One possible solution to improving the breaking process is using a press. The pure Pu metal is formed in a ring shape based on the crucible shape and the anode heel is within this ring. Based on the crucible, these shapes are a predictable size, and the first break could be executed using a punch type press that “pops” the anode heel out of the Pu ring. Then the crucible pieces and/or salt could be removed by hand.

CONCEPT 1:

Small-cont Store Small-mat-1
Med-cont-1 Store Small-mat-2
Med-cont-2 Store Med-mat
 Med-cont-2 Supply Med-mat
 Man Detect Med-mat
Man Move Med-mat
Med-crucible-1 Store Med-mat
 Small-cont Supply Small-mat-1
 Man Detect Small-mat-1
Man Move Small-mat-1
Med-crucible-1 Store Small-mat-1
 Med-cont-1 Supply Small-mat-2
 Man Detect Small-mat-2
Man Move Small-mat-2
Med-crucible-1 Store Small-mat-2
 Furnace-1 Supply Furnace-1-Lid
 Man Detect Furnace-1-Lid
Man Move Furnace-1-Lid
 Man Detect Med-crucible-1
Man Move Med-crucible-1
Furnace-1 Store Med-crucible-1
 Man Detect Furnace-1-Lid
Man Move Furnace-1-Lid
 Furnace-1 Store Furnace-1-Lid
 Furnace-1 Supply Med-crucible-1
 Med-crucible-1 Supply Med-mat
 Med-crucible-1 Supply Small-mat-1
 Med-crucible-1 Supply Small-mat-2
Man Activate Furnace-1
Furnace-1 Create Compos-1
Man Disable Furnace-1
 Med-crucible-1 Store Compos-1
 Furnace-1 Store Med-crucible-1

(cont.)

Furnace-1 Supply Furnace-1-Lid
 Man Detect Furnace-1-Lid
Man Move Furnace-1-Lid
 Furnace-1 Supply Med-crucible-1
 Man Detect Med-crucible-1
Man Move Med-crucible-1
 Med-crucible-1 Supply Compos-1
 Man Detect Compos-1
Man Move Compos-1
 Med-cont-2 Store Compos-1
Man Move Man
Man Grasp Compos-1
 Compos-1 Supply Med-mat
Man Move Chisel
Chisel Separate Med-mat
Man Release Compos-1
 Med-cont-2 Store Med-mat
Man Move Man
Man Grasp Compos-1
 Compos-1 Supply Small-mat-2
Man Move Chisel
Chisel Separate Small-mat-2
Man Release Small-mat-1
 Med-cont-2 Store Small-mat-1
 Med-cont-2 Store Small-mat-2

Figure 5.6: A manual concept generated for the input in Figure 5.5.

CONCEPT 2:

Med-cont-2 Store Small-mat-1
Small-cont Store Small-mat-2
Med-cont-1 Store Med-mat
 Med-cont-1 Supply Med-cont-1-Lid
 Camera-1 Detect Med-cont-1-Lid
SIA5 Move Med-cont-1-Lid
 Med-cont-1 Supply Med-mat
 Camera-1 Detect Med-mat
SIA5 Move Med-mat
Med-crucible-1 Store Med-mat
 Med-cont-2 Supply Small-mat-1
 Camera-1 Detect Small-mat-1
SIA5 Move Small-mat-1
Med-crucible-1 Store Small-mat-1
 Small-cont Supply Small-mat-2
 Camera-1 Detect Small-mat-2
SIA5 Move Small-mat-2
Med-crucible-1 Store Small-mat-2
 Furnace-1 Supply Furnace-1-Lid
 Camera-1 Detect Furnace-1-Lid
SIA5 Move Furnace-1-Lid
 Camera-1 Detect Med-crucible-1
SIA10 Move Med-crucible-1
Furnace-1 Store Med-crucible-1
 Camera-1 Detect Furnace-1-Lid
SIA5 Move Furnace-1-Lid
 Furnace-1 Store Furnace-1-Lid
 Furnace-1 Supply Med-crucible-1
 Med-crucible-1 Supply Med-mat
 Med-crucible-1 Supply Small-mat-1
 Med-crucible-1 Supply Small-mat-2
Man Activate Furnace-1
Furnace-1 Create Compos-1
Man Disable Furnace-1
 Med-crucible-1 Store Compos-1
 Furnace-1 Store Med-crucible-1

(cont.)

Camera-1 Detect Press-1-Door
SIA5 Move Press-1-Door
 Furnace-1 Supply Furnace-1-Lid
 Camera-1 Detect Furnace-1-Lid
SIA5 Move Furnace-1-Lid
 Furnace-1 Supply Med-crucible-1
 Camera-1 Detect Med-crucible-1
SIA10 Move Med-crucible-1
 Med-crucible-1 Supply Compos-1
 Camera-1 Detect Compos-1
SIA10 Move Compos-1
Press-1 Store Compos-1
 Camera-1 Detect Press-1-Door
SIA5 Move Press-1-Door
 Press-1 Supply Compos-1
 Compos-1 Supply Med-mat
Man Activate Press-1
Press-1 Separate Med-mat
Man Disable Press-1
 Press-1 Store Med-mat
 Camera-1 Detect Press-1-Door
SIA5 Move Press-1-Door
 Press-1 Supply Compos-1
 Camera-1 Detect Compos-1
SIA10 Move Compos-1
Med-cont-2 Store Compos-1
Man Move Man
Man Grasp Compos-1
 Compos-1 Supply Small-mat-2
Man Move Chisel
Chisel Separate Small-mat-2
Man Release Small-mat-1
 Med-cont-2 Store Small-mat-1
 Med-cont-2 Store Small-mat-2

Figure 5.7: A concept utilizing robotics for the input in Figure 5.5.

If changes can be made to the furnace and a press is a feasible option for breaking, we can generate a highly automated solution (Figure 5.7). In this workcell, the *SIA5* robot handles loading all three materials into a single crucible and the finer movements of opening the furnace lid and press door. The *SIA10* robot takes care of loading and unloading the apparatuses. This is a feasible option since the *SIA5* is smaller and can maneuver more easily, while the *SIA10* has a higher payload to transport a

variety of objects. A punch press is used to execute the first break, while the *Man* is still required to finish the other material separations.

Instead of first changing the process and/or components to discover potential feasible designs, we could also first ease the requirements, generate *Concept 2*, and then analyze what needs to be modified to allow robotic labor. In this case, we would use very generic components, generate the automated solution, and then search for compatible robot-component pairs to replace the generic ones. The KBS cares more about component types than specific instances of those types because processing differences exists among dissimilar components while different versions of the same component type require similar processing considerations.

Generic function and/or component combinations can also act as a seed for a variety of more concrete sub-function solutions. For example, using a human for a *Detect* function and a robot for a *Move* function could be viewed as tele-operation or the need to *Store* multiple items and individually *Move* them to the same apparatus may invoke the idea of a storage turntable. A single concept could be a starting point for several more detailed designs.

Overall, the KBS can be utilized to not only create new design concepts but suggest modifications to current designs or processes, even if the analyzed designs are infeasible. Implementing component changes in the KBS is straight-forward. If a process change leads to an apparatus change, for instance, properties are simply updated in the component database and the KBS will incorporate this change into its concepts.

5.2.3 Americium Conversion Example

High purity americium-241 can be recovered from plutonium that has been separated from irradiated reactor fuel. When Pu is allowed to “age”, there is an ingrowth of Am-241 from the beta decay of the small fraction of Pu-241 in the plutonium. At LANL, americium-241 is commonly removed from Pu by metal chlorination (formerly molten salt extraction) [LANL, 2008b]. The residues from this and other Pu processes provide an Am-rich feedstream for recovery operations.

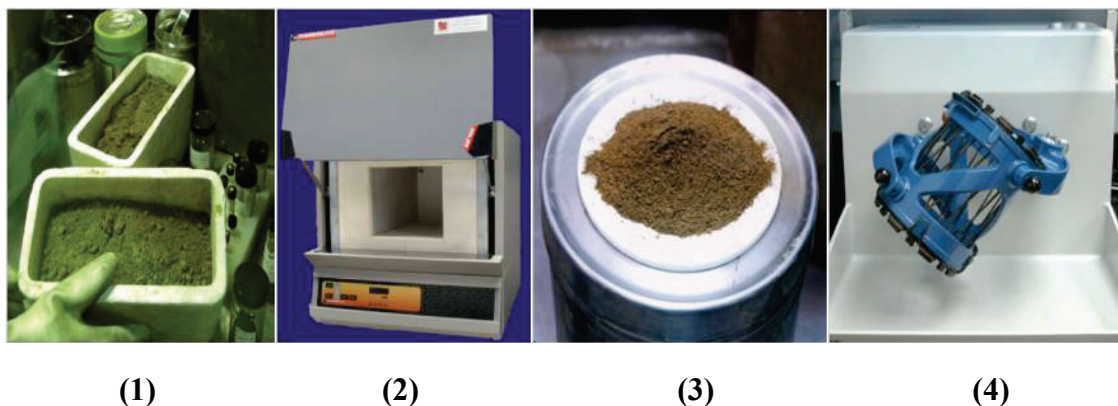


Figure 5.8: Oxide conversion operations pictured for Pu [LANL, 2008b].

The Chloride Extraction and Actinide Recovery (CLEAR) glovebox line is being constructed at LANL for large-scale recovery of Am-241 from Pu processing operations [LANL, 2010]. The Am produced by CLEAR operations is in the form of an oxalate ($\text{Am}_2(\text{C}_2\text{O}_4)_3$). Americium oxalate must be converted to americium oxide (AmO_2), which is the preferred form for distribution to customers. Figure 5.8 shows some steps for the conversion of plutonium oxalate to plutonium dioxide which are also associated with americium conversion. The oxalate (1) is calcined in a furnace (2) to produce a purified oxide (3). A mixing apparatus (4) can be utilized to blend multiple batches together or

create a homogenous single batch. Other glovebox tasks (not pictured) include splitting and/or measuring an oxide to produce a sample for analysis or batch for shipping.

Input:

(Heat of HEAT (input Small-oxa)(output Oxide-1)(noun Small-oxa)(temp 900))

(Mix of MIX (input Oxide-1 Oxide-2)(output Mixed-oxide-1)(noun Mixed-oxide-1))

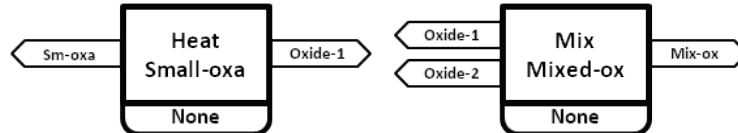


Figure 5.9: The input for americium oxide conversion.

CONCEPT 1:

Med-cont-1 Store Small-oxa
 Med-cont-1 Supply Med-cont-1-Lid
 Man Detect Med-cont-1-Lid
Man Move Med-cont-1-Lid
 Man Detect Med-cont-1
Man Move Med-cont-1
Med-cont-1 Supply Small-oxa
Med-crucible-1 Store Small-oxa
Man Move Med-cont-1
 Man Detect Furnace-2-Door
Man Move Furnace-2-Door
 Man Detect Med-crucible-1
Man Move Med-crucible-1
Furnace-2 Store Med-crucible-1
 Man Detect Furnace-2-Door
Man Move Furnace-2-Door
 Furnace-2 Supply Med-crucible-1
 Med-crucible-1 Supply Small-oxa
Man Activate Furnace-2
Furnace-2 Create Oxide-1
Man Disable Furnace-2
 Med-crucible-1 Store Oxide-1
 Furnace-2 Store Med-crucible-1
Med-cont-2 Store Oxide-2
 Mix-cont-1 Supply Mix-cont-1-Lid
 Man Detect Mix-cont-1-Lid
Man Move Mix-cont-1-Lid
 Man Detect Furnace-2-Door
Man Move Furnace-2-Door
 Furnace-2 Supply Med-crucible-1
 Man Detect Med-crucible-1
Man Move Med-crucible-1

(cont.)

Man Detect Med-crucible-1
Man Move Med-crucible-1
Med-crucible-1 Supply Oxide-1
Mix-cont-1 Store Oxide-1
Man Move Med-crucible-1
 Med-cont-2 Supply Med-cont-2-Lid
 Man Detect Med-cont-2-Lid
Man Move Med-cont-2-Lid
 Man Detect Med-cont-2
Man Move Med-cont-2
Med-cont-2 Supply Oxide-2
Mix-cont-1 Store Oxide-2
Man Move Med-cont-2
 Man Detect Mix-cont-1-Lid
Man Move Mix-cont-1-Lid
 Mix-cont-1 Store Mix-cont-1-Lid
 Man Detect Mixer-1-Lock
Man Move Mixer-1-Lock
 Man Detect Mix-cont-1
Man Move Mix-cont-1
Mixer-1 Store Mix-cont-1
 Man Detect Mixer-1-Lock
Man Move Mixer-1-Lock
 Mixer-1 Supply Mix-cont-1
 Mix-cont-1 Supply Oxide-1
 Mix-cont-1 Supply Oxide-2
Man Activate Mixer-1
Mixer-1 Agitate Mixed-oxide-1
Man Disable Mixer-1
 Mix-cont-1 Store Mixed-oxide-1
 Mixer-1 Store Mix-cont-1

Figure 5.10: A manual design for americium conversion and mixing.

The input for this process is seen in Figure 5.9. The KBS generates the manual processing sequence shown in Figure 5.10. The oxalate is first poured into a crucible. When handling loose objects such as powders, the *Dispense* function is called instead of the *Transport* function. The decomposition of *Dispense* consists of two *Moves*: the move to the pouring location with the pouring motion included and a retracting move. The crucible is then loaded into *Furnace-2*, which is a muffle furnace similar to (2) in Figure 5.8. After the roasting, the furnace is opened and the crucible is removed then set down on the glovebox floor. Both *Oxide-1* and *Oxide-2* are poured from their respective containers into the same mixing container which is then closed. The mixer is opened, the mix container loaded, and then the mixer is closed. The mixer runs and blends the oxides together to form a single mixed oxide. If one oxide is uranium and the other is plutonium, then the mixing produces a mixed oxide similar to that used in MOX fuel pellets.

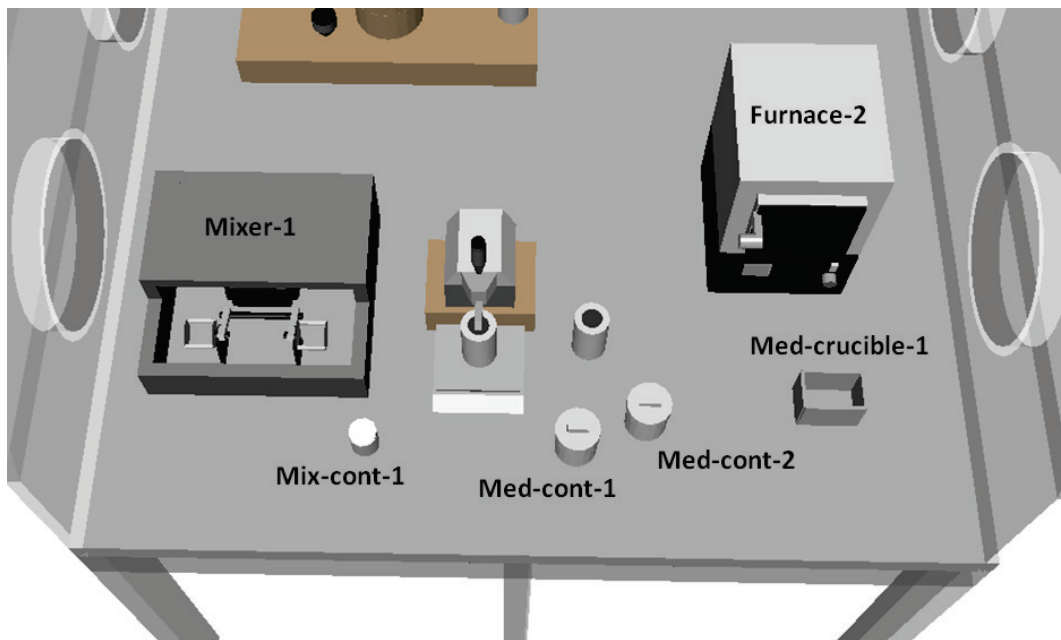


Figure 5.11: An example layout for *Concept 1* in Figure 5.10.

This example demonstrates some operational sequencing currently built into the rules. For instance, after roasting, the crucible contents (*Oxide-1*) need to be transferred to a mixing container. Instead of removing the crucible from the furnace and pouring the contents in the mix container in a single move, the crucible is first removed from the furnace and set down on the glovebox floor. Then, *Oxide-1* is poured into the mix container. The ability to choose either processing option may be desired. However, the selected operation sequence presents the general operations that can be modified to produce other (simpler) scenarios. Also, the KBS does not currently handle partial material pours that leave the same material in different containers. This functionality can be built in, but demonstrates that there can often be new operational scenarios identified when addressing a new design problem.

5.2.4 Welding Example

The Advanced Recovery and Integrated Extraction System (ARIES) glovebox line at LANL takes surplus plutonium and converts it to an oxide that can be used to create MOX fuel for commercial reactors. The ARIES line disassembles a plutonium pit, converts the Pu metal to an oxide, mills, blends, and performs physical and chemical analyses on the oxide, and then packages it for long-term storage [LANL, 2008a]. This process is performed using eight modules.

Some of these tasks have been described in previous examples. For this example, we focus on the automated welding used in the packaging ARIES glovebox. The system first receives a convenience can containing fissile material (Figure 5.12). A robot then places the convenience can inside a DOE standard inner can. This inner can is then loaded into an automatic welding system and a lid is welded on. After welding, the inner

can is transferred to a decontamination chamber and is then run through a series of radiation surveys.

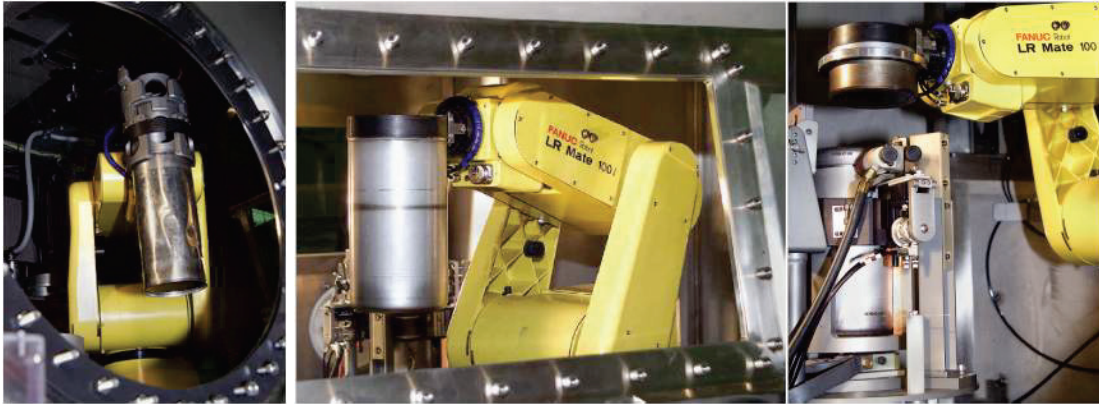


Figure 5.12: The robot handling a convenience can (left), holding a DOE standard inner can (center), and positioned next to the automatic welder (right) [LANL, 2008a].

Input:

(Store of STORE (input Med-cont-1 Small-cont)(output Med-cont-1)(noun Small-cont))
(Weld of WELD (input Med-cont-1 Med-cont-1-Lid)(output Assembly-1)(noun Assembly-1))

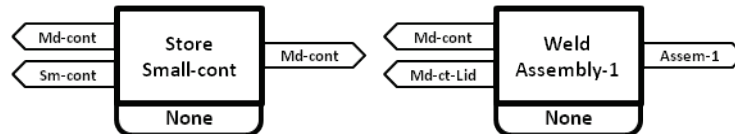


Figure 5.13: Input for placing a convenience can inside another can then welding the outer can.

The input functions for the welding process are listed in Figure 5.13. The task plan generated for the robot is seen in Figure 5.14. The *Store* function is used to force the loading of the convenience can (*Small-cont*) into the inner can (*Med-cont-1*). In the ARIES line, containers are received and transported to fixed locations every time.

Concept 1 includes *Camera-1* for detecting the locations of containers in a dynamic environment or to create a more flexible system.

CONCEPT 1:	Rules 1:
Med-cont-1 Supply Med-cont-1-Lid	EMBODY-store
Camera-1 Detect Med-cont-1-Lid	add-OPEN-load
SIA5 Move Med-cont-1-Lid	dec-OPEN
Camera-1 Detect Small-cont	EMBODY-detect
SIA5 Move Small-cont	EMBODY-move
Med-cont-1 Store Small-cont	dec-LOAD-single
Camera-1 Detect Med-cont-1	dec-TRANSPORT
SIA5 Move Med-cont-1	EMBODY-detect
Welder Store Med-cont-1	EMBODY-move
Camera-1 Detect Med-cont-1-Lid	EMBODY-weld
SIA5 Move Med-cont-1-Lid	add-LOAD-connect
Welder Store Med-cont-1-Lid	dec-LOAD-single
Welder Supply Med-cont-1	dec-TRANSPORT
Welder Supply Med-cont-1-Lid	EMBODY-detect
Welder Join Assembly-1	EMBODY-move
Welder Store Assembly-1	dec-LOAD-single
	dec-TRANSPORT
	EMBODY-detect
	EMBODY-move
	dec-WELD-single

Figure 5.14: A generated task plan for a robot performing packaging in the ARIES line.

Another example generates a concept for a robotic welding workcell with the task plan shown in Figure 5.15. The *SIA10* robot welds three parts together to form an assembly. Each part is placed into a fixture before it is welded and the KBS tracks flow locations using the *inside* and *contains* slots. When *Part-1* and *Part-2* are welded, they are contained in different fixtures, so the resulting *Assembly-1* is also located inside multiple containers. At this conceptual level, the precise welding path is not specified nor is the exact positioning of the fixtures and parts within them. This information would be added during detail design. The concept assumes the *SIA10* robot is capable of opening fixture locks and providing the needed welding accuracy and repeatability.

CONCEPT 2:

Med-cont-1 Store Part-1
Med-cont-2 Store Part-2
Large-cont Store Part-3
 Camera-1 Detect Fixture-1-Lock
SIA10 Move Fixture-1-Lock
 Med-cont-1 Supply Part-1
 Camera-1 Detect Part-1
SIA10 Move Part-1
Fixture-1 Store Part-1
 Camera-1 Detect Fixture-1-Lock
SIA10 Move Fixture-1-Lock
 Med-cont-2 Supply Part-2
 Camera-1 Detect Part-2
SIA10 Move Part-2
Fixture-3 Store Part-2
 Fixture-1 Supply Part-1
 Fixture-1 Support Part-1
 Fixture-3 Supply Part-2
 Fixture-3 Support Part-2
SIA10 Move Weld-torch
Weld-torch Join Assembly-1

(cont.)

Fixture-3 Store Assembly-1
 Fixture-1 Store Assembly-1
 Camera-1 Detect Fixture-1-Lock
SIA10 Move Fixture-1-Lock
 Fixture-1 Supply Assembly-1
 Fixture-3 Supply Assembly-1
 Camera-1 Detect Assembly-1
SIA10 Move Assembly-1
Fixture-2 Store Assembly-1
 Large-cont Supply Part-3
 Camera-1 Detect Part-3
SIA10 Move Part-3
Fixture-3 Store Part-3
 Fixture-2 Supply Assembly-1
 Fixture-2 Support Assembly-1
 Fixture-3 Supply Part-3
 Fixture-3 Support Part-3
SIA10 Move Weld-torch
Weld-torch Join Assembly-1
 Fixture-3 Store Assembly-1
 Fixture-2 Store Assembly-1

Figure 5.15: A concept for a robotic automated welding workcell.

Welding functional requirements can be defined using the *accuracy*, *repeat*, and *velocity* slots to filter between welding apparatuses and operators. Additionally, *dexterity* and *motion* slots are checked if a considered embodiment involves tool use by a human or robot.

5.3 DISCUSSION

The KBS is essentially a shell for manufacturing system design generation in the same sense that CLIPS is a shell for knowledge-base system development. The KBS contains all the supporting functions and rules for material handling and component operation (ex: open, close, load, and unload) for user-defined specific processing functions (ex: weld, break, and heat) and database components. That is, the user develops their database and inputs process functions and the KBS fills in the needed components

and processing tasks. The knowledge-base framework is flexible and can support extensions as needed.

5.3.1 Concept Generation

To demonstrate the use of this shell, we generated concepts for specific processes at LANL. These results emphasize how the KBS framework can successfully produce conceptual designs. At this design stage, many details are unknown, so we focus on the general compatibility/feasibility of concepts using general requirements that can help filter results if desired.

Our design concepts consist of two intertwined parts: the workcell contents (necessary components) and the processing instructions for the workcell. As a baseline, the KBS can duplicate current designs and their processing nature (which is mainly manual). The KBS also generates designs with increased automation. The whole range of concepts from manual to hybrid (human and robotic) to fully automated (all robotic) can be captured by relaxing requirements and including particular database components. Actual concept feasibility must be analyzed more in-depth by designers as it relies on more specific information about the processes and the proposed system components. In particular, robotic feasibility depends on whether the processing environment is static and/or dynamic, how well-defined the tasks are, and the exact nature of apparatus/component manipulations for operators.

The KBS generates all possibilities for the provided database. Depending on the active database components and requirements, a few to thousands of options can be generated. Although there can be hundreds to thousands of concepts, only a set number of *task sequence types* can be identified. These are typically related to the number of high-

level functions and the possible combinations of *component types* (operators, apparatuses, containers, and tools) for those functions. For example, if two *Break* functions exist, operations may proceed from a press to manual chiseling where the human grasps or a fixture stabilizes or operations may proceed in the opposite direction. Thus, the processing operations could be from an apparatus to an operator grasp or from an apparatus to a fixture or from a fixture to an apparatus, etc. Additional sequence variations stem from the states of those components, such as whether they are opened or closed or inside another component. This generates various insertion location combinations for open, close, unload, and load functions in the main task sequence. All these contributions result in a combinational increase in *processing sequence types*. For example, between Figure 5.2 and Figure 5.3 there are four (of many) unique task sequence types shown.

Enumerating these sequences is important for concept generation. Once we have the sequence types, which correspond to certain generic embodiments, we can switch in similar components that do not change the processing needs but can still fulfill the functions. This allows for another combinatorial increase based on the number of instances of a certain component type in the database. For instance, sequences generated using a generic robotic manipulator may be compatible with both the *SIA5* and *SIA10* robots, doubling the number of designs each time both robots can embody a function in the task sequence. The KBS increases designs in both ways: the component database enumerates components for embodiment and the rules help enumerate processing sequence types based on embodiments. Typically, the KBS selects components (i.e. enumerates embodiments) then determines the necessary processing (i.e. establishes a task sequence type).

Thus, our accomplishment is not only the number of workcells generated with different component sets, but the number of concepts with different processing sequences. Without automated design, some concept types may be overlooked since there are many details to track that affect embodiment and processing needs. The designer only has to include component types of interest and the KBS automatically enumerates *design types*.

5.3.2 Task/Process Planning

In our concept generation technique, we have imbedded a generative process planning algorithm where a new plan is constructed for each sub-process based on the components, operators, materials, and their states – there is no database of pre-defined process plans. This helps produce more creative design concepts. Many process planning operations and robotic task plans start with a high-level description of a part or product to manufacture [Rocha, 1997] and decompose it into task primitives based on product parameters and available manufacturing resources (machines, tools, etc). Our plan generations begin with the high-level operations the manufacturing system as a whole must perform and these are decomposed into primitives that could be the high-level starting point for product manufacturing plans. Therefore, we evaluate our process plans by their ability to represent and communicate information at our desired level of abstraction since direct comparison with more concrete process plans is not reasonable.

In relation to task planning for design and manufacture, we can evaluate our KBS relative to the general areas that a successful planner must address (adapted from [Gottschlich, 1994][Kiritsis, 1995][Rocha, 1997] and [Rosell, 2003]). The implemented KBS utilizing FBS modeling handles these various facets at the conceptual level:

- *Developing a task representation* – The common language is constructed so its terms form a type of basis. A single functional term can apply to various embodiment types: human, robot, apparatus, container, etc. The generic nature of functions and components reflects the conceptual state of knowledge.
- *Defining the required operations* – The user inputs the required functions and the rules fill in additional functions to create a continuous task sequence.
- *Establishing the necessary workcell resources* – The component database can contain as much or as little labor, machine, tool, etc, options as desired.
- *Incorporating the precedence of operations and handling multiple ways to execute sets of tasks* – The rule coding generates a single sequence among multiple sequences when multiple tasks could be performed in parallel. The rule conditions force a particular sequence. No evaluation is performed to select the “best” available option.
- *Utilizing form, connectivity, and/or adjacency information* – Generic form information is utilized for requirements and task compatibility. Component slots track the movement and relative location of flows in the workcell which directly affects the validity of a particular task sequence.

Our task plan descriptions are generic but understandable by a human designer. The task plan is derived from the full solution so the file containing it can provide additional information such as where materials are being moved from and to, if desired. The user does need to understand how functions are formulated and what particular verb-noun combinations signify. As demonstrated in the next chapter, the full solution file can be interpreted by the computer to perform a layout. The layout plus the geometric details in the component database make it easier for a human to visualize a design.

5.3.3 Limitations and Issues

The KBS cannot enumerate all possible task sequences for a given workcell concept. As previously described, when parallel tasks exist in a sequence, a single task order is selected based on the coding in the rules. This does not affect whether a particular combination of workcell components is feasible, but it would affect performance metrics for workcell layout since certain task orders could be more efficient, for instance. Rule-base modifications could allow more generative aspects such as presenting the multiple sequences derived from parallel tasks or adding extra functions to decompositions to possibly introduce interesting embodiment combinations. Other opportunities to incorporate creativity will be further described in Chapter 10.

Not all possible instance states are currently supported, but they can be added. The *inside* slot only stores the flow another flow is immediately inside. In the rules, the KBS only checks for two levels of “inside” when considering opening and closing operations. Thus, the KBS will detect if a material is inside a container that is inside an apparatus, but not any further. Also, only assemblies can have multiple values in their *inside* slots. This means the KBS cannot handle other flows contained by multiple containers or apparatuses. This would affect operations such as pouring a portion of a powder into another container, but the rule conditional tests could be altered to manage these scenarios. The rules also do not check if an operator is busy with another task or requires a tool change between tasks. With continued use and implementation, the KBS will support more and more processing scenarios.

Many generated designs are inefficient such as ones that unnecessarily switch between operators for adjacent tasks or use a camera to detect an object moved by a human. Sorting through designs can be tedious and design filtering would be useful. Also, since some requirements and components are generalized, *final* compatibility and

design feasibility cannot be determined at this stage. This is mainly due to the lack of detailed geometric information which is characteristic of conceptual design. However, as discussed, generating infeasible concepts can be beneficial to motivate design changes. In practice, designers' would review and evaluate the generated designs which have passed the rough automated feasibility check. Thus, generated concepts provide a set of options to evaluate for inclusion in future design phases.

CLIPS provides verification and validation checks including static and dynamic constraint checking of slot values and function arguments. The static checking detects errors when the CLIPS code complies. The slot value attributes can be constrained to a particular data type, range, default value, etc, to detect or prevent inconsistencies in instance declarations. The function and flow classes utilize such constraints in some of their slots. For example, errors in database component slots are detected when the database is loaded. Implementing input function I/O and noun checks would be beneficial, but their accepted values may have to evolve with the knowledge-base.

The dynamic constraint checking will catch errors during generation, which stops the program. When an error is flagged, it may be a few iteration cycles from when the logical error occurred, making identification of the code producing the disallowed slot value difficult. Most errors that prematurely terminate concept generation result from miscues in rule application. This can result when multiple add and/or decomposition rules fire at the same time or no applicable rules are found because the rules do not match the current instance states. The latter case can stem from a lack of knowledge in the knowledge-base or incomplete information in function or component instances. If the user understands how to define function instances and what component/flow types should be included, then errors should be minimal barring any new processing scenarios the

knowledge-base has not yet encountered. More error checking would be needed for a more commercial-ready product.

5.4 SUMMARY

Many expert systems are very specific in their application domain. Being conceptual in nature, the KBS is developed to maintain generality and spur design creativity. The knowledge base stores previous design information, but that information is applied in such a manner to avoid specific design solutions.

The term “knowledge-based system” is used rather than “expert system” because new information can always be added. The open-endedness of our concept generation program reflects this, and indeed, in our presented examples, we had to set the boundary of incorporated knowledge to a particular realm. Our accomplishment is the creation of an all-inclusive, adaptable concept generation framework that supports new knowledge and design problems. This framework includes a powerful design representation that includes a workcell’s function structure, configuration, and operational task sequence. In fact, structural and operational information are concurrently addressed in design generation.

In the next two chapters, we will describe the concept layout procedure. We can apply approximate evaluation metrics at this stage to give preference to generated concepts. We can also compare optimized layouts to those produced manually from previous work on LANL material handling systems. Through the sum of this work, we demonstrate how our system (KBS plus optimization modules) emulates a designer’s conceptual design of glovebox manufacturing systems in the nuclear domain. This is in regard to component and labor selection, procedure development, and glovebox layout.

Chapter 6: General Layout Approach

This chapter outlines our technique to find preliminary layouts for glovebox systems. After concept generation, the designer manually selects designs for continued development. Once we configure these designs, we can further evaluate feasibility and preliminary metrics of performance. This evaluation is still conceptual in nature, although some rough estimates for component geometry are required. In contrast, during detailed design, we have a CAD model and analysis software (for robotic simulation, dose calculations, etc) to better evaluate layout performance. However, even at this level of abstraction, we can make reasonable estimates of feasibility and begin to rank design concepts.

The KBS generates concepts that include the components/resources utilized by the glovebox workcell and a suggested task plan for executing the process using those resources (which addresses connectivity). In theory, this concept is valid since all sub-tasks are identified and assigned to a particular component or operator. However, spatial and geometric constraints may prevent a concept from having a feasible layout or lead to a poor layout. Facility constraints may require that the system fit within an existing glovebox and/or all the components that an operator needs to execute its assigned functions may not fit within its workspace. Also, even if a valid configuration exists, it may take a considerable amount of time to identify. Thus, we employ optimization techniques to build and evaluate valid workcell layouts.

6.1 PREVIOUS WORK

The layout problem takes many forms and has many areas of application. System configuration often refers to how system components are located relative to one another. Commonly, the designer must also make decisions about how components are connected, or the design topology. Optimization techniques can be applied to guide the layout process. We will describe relevant research threads at a high level and then in later sections more specifically describe how these optimization formulations and techniques led to our layout approach.

6.1.1 Facility and Machine Layout

Researchers have often considered the facility layout problem [Singh, 2006][Drira, 2007]. This problem identifies the best arrangement of facilities among a number of locations or a defined space while minimizing objectives such as time, cost, traveling distance, and/or flows. Drira [2007] lists different attributes, formulations, applications, and solution techniques for facility layout problems. The facility layout problem is commonly formulated as a Quadratic Assignment Problem (QAP), which is NP-complete [Singh, 2006]. Other approaches use graph theory models or formulate the problem as a Mixed-Integer Problem (MIP) when incorporating continuous variables. Exact methods such as branch and bound can be applied but typically only small-sized problems can be solved optimally [Drira, 2007]. For large-sized problems, metaheuristic approaches are commonly employed and include genetic algorithms, Tabu search, simulated annealing, and ant colony algorithms [Singh, 2006][Drira, 2007]. Other heuristic approaches include construction and improvement algorithms. Construction algorithms build a complete layout from scratch, whereas improvement algorithms start with an initial layout and improve upon it.

Layout can also be performed for each facility or cell which is the intra-cell layout problem. This problem can include determining the grouping of machines, the type of material handling system, the type of machine layout, and the location of machines inside the cell/about the material handling path [Heragu, 1989][Hamann, 1992][Drira, 2007]. Types of material handling systems include robotic manipulators, gantry robots, Automated Guided Vehicles (AGVs), conveyors, and carts, while layout types include single, double, or multiple row, loop/circle, and open-field/free [Hamann, 1992][Souilah, 1995][Drira, 2007]. Both inter-cell and intra-cell problems are closely related in formulation and solution techniques and can represent a number of real-world design problems.

6.1.2 Component Layout

Optimal configuration of mechanical assemblies is also a related research area for layout design. Component layout or packing problems are commonly addressed in design and can involve either 2-D or 3-D layouts [Dowsland, 1992][Cagan, 2002]. In regard to mechanical and electro-mechanical design [Cagan, 2002], difficulty in layout automation stems from:

- 1) The modeling of design objectives and constraints,
- 2) The efficient calculation of objectives and constraints, and
- 3) The identification of appropriate optimization search strategies.

A variety of techniques address these difficulties for real design problems. Szykman [1997] utilized a simulated annealing algorithm with penalty functions for constraint violations to optimize the layout of power drill components. Campbell [1997] developed a quicker heat transfer analysis approach that could be incorporated into a

simulated annealing algorithm to achieve high packing densities for electronic component layouts while avoiding overheating. Grignon [1998] developed a method for optimizing complex mechanical assemblies considering the placement and orientation of freeform components. The method applied a modified GA to the configuration of components in a satellite and a car engine based on maximum compactness or accessibility. Yi [2008] utilized a “packing GA” to maximize vehicle performance according to three competing objectives: maintainability, survivability, and vehicle dynamic performance. Yin [2000] used an extended pattern search algorithm applied to packing problems, car engine configuration, and concurrent heat pump layout and routing. Pattern search algorithmic extensions included random component picking orders, occasional step jumps, and component swapping. Hybrid methods have also been developed [Cagan, 2002]. Previous component layout work identifies modeling techniques and optimization algorithm modifications that may help build a more effective layout approach for our design problem.

6.1.3 Robot and Manufacturing Workcells

The areas of facility and component layout provide more general factors and techniques for design layout. More specifically, there is also work related to the positioning of robots for tasks and the layout of robotic manufacturing workcells.

Much research has addressed optimizing robot task completion. This includes establishing greater task compatibility utilizing measures for determining favorable manipulator postures/joint configurations for effecting or controlling velocity or force at the end-effector [Yoshikawa, 1985][Chiu, 1988][Dubey, 1988] or general kinematic performance about a task point [Klein, 1985][Pamanes, 1991]. Criteria can also be used

to evaluate robot performance during manipulation [Pholsiri, 2004][Tisius, 2009]. LeGoullon [1997] used criteria-based decision making to quantitatively compare the quality of different robot workcell configurations for airframe assembly.

Other work has dealt with selecting the best position of the manipulator base relative to task locations. Abdel-Malek [2004] formulated an augmented Jacobian to find a locally optimal location for the manipulator base in the workspace that takes into account singular behavior and joint limits. Zacharias [2007] developed the capability map to visualize a robot's kinematic capability in the workspace and identify workspace locations that are easy to reach and/or provide versatile manipulation. Williams [2010] used a 2-D variation of the capability map, the task plane, to spatially locate manipulation targets fixed with respect to each other in more dexterous workspace regions.

Other layout methods optimize based on performance metrics that help reduce the cycle time of operations and/or the distance traveled by the manipulator. Lueth [1992] proposed a technique for the automatic planning of layouts in three-dimensional space using the Cartesian configuration space. In this work, layouts are optimized with regard to the robot movements and flow of material. Tay [1996] utilized heuristics to select the order of placement and locations of stations around a manipulator based on the station interactions during operation and the horizontal distances between their access points. Mata [1998] used the travel time and the total joint displacement for all possible sequences of operations required by the robot at a given period of time for evaluation.

Without detailed manipulator models, work involving quantitative robot performance metrics is beyond the scope of our work. Furthermore, we do not know enough about the task points for these metrics to be useful. However, the limiting kinematic considerations (ex: joint limits, singularities, reachable and dexterous workspaces, etc) that affect these metrics and manipulation abilities in general could be

incorporated to generate rough estimates of operator-task performance, if desired. Metrics involving time and/or distance at a system level will be more relevant for determining the efficiency of a manufacturing workcell layout.

6.2 GENERAL PROBLEM ELEMENTS

In this section, we give a brief overview of the nature and needs of our optimization problem. Specific design representation information and optimization modeling will be described more in-depth in later chapters.

6.2.1 Mathematical Model

This problem is multi-disciplinary and benefits from aspects of all problem types discussed in Section 6.1. Our layout problem can utilize the mathematical programming model frequently used for design optimization [Papalambros, 1995]:

$$\begin{aligned}
 &\text{minimize} && f(\mathbf{x}, \mathbf{p}) \\
 &\text{subject to} && \mathbf{g}(\mathbf{x}, \mathbf{p}) \leq 0 \\
 &&& \mathbf{h}(\mathbf{x}, \mathbf{p}) = 0
 \end{aligned} \tag{6.1}$$

where f is the scalar objective function, \mathbf{g} is the vector of inequality constraints, and \mathbf{h} is the vector of equality constraints. In mechanical design, these functions are generally nonlinear and thus the model represents a non-linear programming (NLP) problem [Papalambros, 1995]. These are all functions of the design variable vector \mathbf{x} and parameter vector \mathbf{p} . Our design variables include component location (x , y , z) and orientation (θ_x , θ_y , θ_z) which represent rotations about the global x , y , and z axes.

Depending on the formulation, these variables could be discrete or continuous. Component parameters include geometric dimensions and access points. Since database components are generic, these parameters represent approximations of expected qualities.

6.2.2 Objectives

With considerations from multiple fields, our layout problem is inherently multi-objective. As discussed, the optimization does not focus on robot task optimization and the associated performance metrics. We are more concerned with performance at a system level for the application domain, particularly manufacturing and glovebox work. The main factors for workcell material handling include total process time and distance traveled during operation. Additional considerations for evaluating the quality of a *glovebox system* include:

- Dose: whole body and extremity
- Ergonomics
- Safety/hazards: thermal, chemical, mechanical, sharps, etc
- Maintainability/accessibility
- Waste generation

We will focus on two metrics for better visualization of results: dose and process/cycle time. These can be competing objectives. For instance, minimizing work time suggests that components be placed near to each other and operators (high density), while dose minimization encourages worker separation from locations where radioactive material resides (low density). We should see the effects of these two objectives in our results.

In nuclear operations, especially in small batch and low throughput operations, radiation protection is more important than production. Thus, we should be able to

describe the relative preference of these objectives *a priori*. A weighed sum approach to scalarizing our objective function should be sufficient.

The objective function presents mathematical challenges to optimization as it will likely contain many local minima and be discontinuous. There is no continuous analytical function relating component locations and orientations to time. The dose calculation is also complex although analytical dose formulas exist. In simple models, dose has a nonlinear relationship to separation distance between the human and the radioactive material. Also, there is time dependence that must be considered. Thus, based on our problem, gradient-based techniques requiring differentiable functions will not apply. Objective function implementation and calculations will be described in Section 7.2.

6.2.3 Constraints

There are two main constraint sources. Facility and component layout problems incorporate the following constraints:

- No component overlap (or minimum separation distance)
- Components fit inside designated container or space
- Proximity or adjacency (desired or not desired)
- Prohibited locations

Additional constraints are related to robotic/human manufacturing workcells:

- Collision-free reachability of component access points
- Collision-free transport path/trajectory

The constraints are generally numeric, but analytical functions in terms of the design variables do not exist.

Many component layout problems are formulated as unconstrained optimization problems. In the constrained case, dealing with constraints directly requires generating very large numbers of *feasible* designs which greatly increases computational time. However, our design search would benefit from allowing component overlaps/collisions as better and collision free locations may be a small step away from the current infeasible layout. To convert the problem to an unconstrained optimization problem, we can develop penalties for constraint violations to add to the objective function. These penalties are relative to component overlap, operator reachability, path collisions, and boundary violations. Including penalties, the transformed/penalized objective function f_p becomes

$$f_p(\mathbf{x}, \mathbf{p}) = f(\mathbf{x}, \mathbf{p}) + P(\mathbf{h}, \mathbf{g}, r) \quad (6.2)$$

using the penalty function P . The penalty function can take different forms [Papalambros, 2000][Arora, 2012]. A general formulation is

$$P(\mathbf{h}, \mathbf{g}, r) = r \left[\sum_{i=1}^j [\max(0, \mathbf{g}_i(\mathbf{x}, \mathbf{p}))]^n + \sum_{i=1}^k [\mathbf{h}_i(\mathbf{x}, \mathbf{p})]^n \right] \quad (6.3)$$

for each constraint i where $r > 0$ is a scalar parameter and n is an integer, commonly 1 or 2. The *quadratic loss function* is defined by $n = 2$. In our problem, we will only penalize inequality constraints, so the second term drops out of our penalty formulation. Equality constraints will be directly enforced/propagated. In addition to r , the proper weighting and/or normalization factors also need to be added to the penalty function. As this is a

minimization problem, assessing positive penalties should force the algorithm to converge on a solution that has no constraint violations.

6.3 SEARCH MECHANISM

As seen in Section 6.1, previous layout techniques have utilized various algorithms based on the properties and goals of the design problem. This information and the general applicability and benefits of various techniques can help guide our selection of an optimization engine.

6.3.1 Applicable Algorithms

Cagan [2002] defines the *layout space* as “the mathematical representation of the space of configurations mapped against cost per configuration.” Given the nature of our objective functions, this space is typically non-linear and multi-modal. In these cases, “deterministic algorithms are unable to navigate such a space for globally near-optimal solutions, and stochastic algorithms are usually required for solutions of good quality” [Cagan, 2002]. Furthermore, many design problems require modeling with discrete variables which leads to combinatorial optimization formulations that are basically unsolvable without heuristics [Papalambros, 1995].

Thus, metaheuristic or stochastic techniques such as simulated annealing (SA) and genetic algorithms (GAs) are highly applicable to our problem for these reasons and others that have been described. Many of these strategies were utilized in the examples presented in Section 6.1. These methods “are attractive when functions are discontinuous or nondifferentiable but inexpensive to compute, good starting points are known, and many local optima may exist” [Papalambros, 1995].

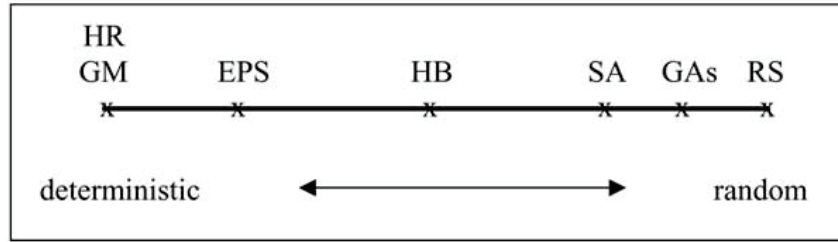


Figure 6.1: Computational approaches to layout problems on a continuum from deterministic to random [Cagan, 2002].

Although stochastic principles are needed for global optimization, the algorithmic implementation does not need to be exceedingly random. In the vein of our knowledge-based concept generation approach, we can apply heuristics to layout design for a more efficient and deterministic search. The applicability and effectiveness of heuristics will help drive where the algorithm is located between deterministic and random (see Figure 6.1). On one end are heuristic rule-based (HR) and gradient-based (GM) deterministic methods and on the other end are stochastic methods such as random search (RS). Extended pattern search (EPS), hybrid methods (HB), SA, and GAs are between these end points.

6.3.2 Algorithm Considerations

Our goal is to utilize established techniques to develop an algorithm that generates reasonable layouts. Since these designs are still conceptual in nature, we are more interested in the structure of the configured layouts to make qualitative and perhaps quantitative comparisons and/or conclusions concerning task allocation (human or robotic) in the workcell(s). We also want an algorithm that is generally applicable to a

wide variety of problems and can be used to quickly configure many concepts, if desired. Considering our design problem, a hybrid method using principles from stochastic and deterministic methods may be very suitable for our needs.

6.3.2.1 Stochastic Approaches

GAs and SA are particularly useful for global optimization due to their ability to avoid local minima. Stochastic mechanisms such as crossover and mutation and acceptance of “uphill” moves, respectively, help accomplish this task. However, these searches can be quite large. Simulated annealing, for instance, can take on the order of 100,000 total iterations in its search for a solution [Campbell, 1995][Szykman, 1997]. GAs also require a significant amount of calculation even for reasonably-sized problems and sometimes parallel computing is utilized to speed up processing [Arora, 2012]. Furthermore, both SA and GAs do not guarantee converge on the global optimum [Szykman, 1997][Arora, 2012], but they do provide near-optimal solutions.

Although not extremely difficult to implement, these algorithms can be difficult to tune. Small changes in the evaluation function weighting factors can make a big difference in the results of a genetic algorithm [Cagan, 2000]. Finding “good” cooling schedules or other parameters for SA algorithms can be difficult as they typically depend on the specific problem.

In our case, it may not be worth spending so much computational effort and time on a design that is still quite conceptual in nature. The global optimum is probably not needed, but we still desire a “good” solution. Such solutions provide a feasible and reasonable layout with enough structure to make evaluations about future design choices.

The benefits of stochastic techniques are desirable, but not the increased computational time. Thus, we consider moving closer to the deterministic end than these methods.

6.3.2.2 *Deterministic Approaches*

For our problem, the domain heuristics that exist are related to directions that move a design toward *feasibility*. In particular, this includes moving components toward the operator's reachable workspace, adjusting component access location orientations, or working from a single gloveport if possible. The influence of such moves on the objective function is non-trivial. If possible, structured application of the best available moves for a given configuration would lead to a more effective search.

Direct search methods are applicable in this case. They are deterministic and sequentially examine trial solutions to compare with the current “best” solution and have a strategy for determining the next trial solution [Cagan, 2000]. Pattern search methods are a subset of direct search methods and are *gradient-related* techniques that employ heuristics to identify a direction of “steep” descent [Torczon, 1997a]. Pattern search employs a pre-defined pattern (Figure 6.2) to systematically make exploratory moves for the variables one at a time, evaluating the new configuration with the current best one, and accepting the new move only if it is better. The exploratory move steps are decreased as the algorithm converges on a solution.

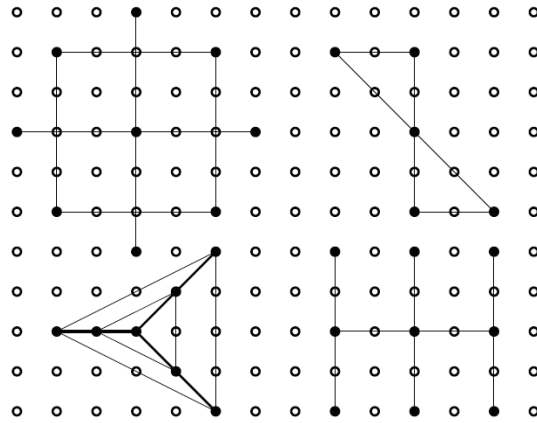


Figure 6.2: Some example patterns and their mapping onto a lattice [Torczon, 1997a].

Torczon [1997a] suggests that pattern search methods are often useful in the following situations:

- 1) Evaluation of the objective function f is inaccurate
- 2) The derivatives of f are either not available or not reliable
- 3) The function f is not smooth
- 4) Only ordinal information about function values is available

Our problem is suitable for pattern search. Using such an algorithm, we do not have to be so precise, which fits the nature of conceptual design. Additionally, pattern search methods are good at locating the general region of a stationary point and can find a “quick and dirty” solution to a problem [Torczon, 1997a]. At a basic level, to do this, we only need to control the step size to ensure that the algorithm does not converge too quickly. More information on the principles and convergence of pattern search can be found in [Torczon, 1997b][Lewis, 1998].

6.3.2.3 Extended Pattern Search

Extended pattern search (EPS) [Yin, 2000][Aladahalli, 2007] is a hybrid method that adds stochastic principles to the general pattern search approach. For layout design, stochastic additions include randomized component placement order, occasional “step-jumps”, and component swaps [Yin, 2000]. Souilah [1995] also utilized component swapping, but in a SA layout algorithm. Extended pattern search can generate equivalent quality layout solutions in a time that is an order of magnitude or two less than SA [Cagan, 2000][Yin, 2000].

Based on the above information, we develop an extended pattern search algorithm for the following reasons:

- Applicability to our mathematical model (objective function, constraints, solution space, etc) and the precision of known information,
- Easier adjustment of control parameters,
- More effective search of the design space to decrease computational time, and
- Global behavior.

Existing EPS work for component layout provides a framework for developing our algorithm, while the specific needs of our problem will drive new extensions.

6.4 SUMMARY

In this chapter, we gave an overview of previous layout optimization work relevant to our problem. Based on the nature of our design problem and the trade-offs of various optimization engines, we will develop an extended pattern search algorithm to generate workcell layouts. In the next chapter, we describe how the general model described in Section 6.2 was applied to our design problem. In particular, we will establish the specific mathematical model and the derivation of objective function and

constraint calculations. The implementation of the extended pattern search algorithm is discussed in Chapter 8. Assumptions and simplifications related to our scope will be described as necessary.

Chapter 7: Layout Optimization Model

The layout optimization algorithm is constructed to configure designs produced by the concept generation algorithm. The designs are given as an ordered sequence of functions (described using a string representing a CLIPS instance) with embodiments listed in the *component* slot. The function components are those that need to be configured in the workcell. The goal for workcell layout is to take a design description, find an initial layout, and then optimize it. At the conceptual level, a layout represents the general form a design should take utilizing the given task sequence, task allocation, and generic components.

7.1 REPRESENTATION

The workcell layout optimization is coded and executed in CLIPS. The CLIPS function and component instances allow for easy storage and recall of information. Also, we can utilize many of the existing built-in CLIPS functions to query and update instance states, although there is some amount of overhead to using CLIPS (built in C) rather than a high-level language such as C++. Using CLIPS also allows for future extensions that may incorporate rules into optimization to provide further generation features (see Chapter 10).

The solution files from concept generation can be fed directly to the optimization algorithm. Thus, we directly utilize the concept generation design representation (i.e. task plan). The optimized layout is represented by the positions and orientations of all the components utilized in the task plan. The origins for all *Moves* are also defined.

7.1.1 Component Geometry

Each component has a rough geometry. There are no CAD models, so each component is represented by either a box or cylindrical shape. Boxes have a length, width, and height triple while cylinders have a radius and height designation. These parameters are stored in the component's *dim* slot. The component's local origin is at the center of its base (see Figure 7.1). A *component location* defines where the component's local origin/center is located relative to the global origin, which is typically a corner of the glovebox floor.

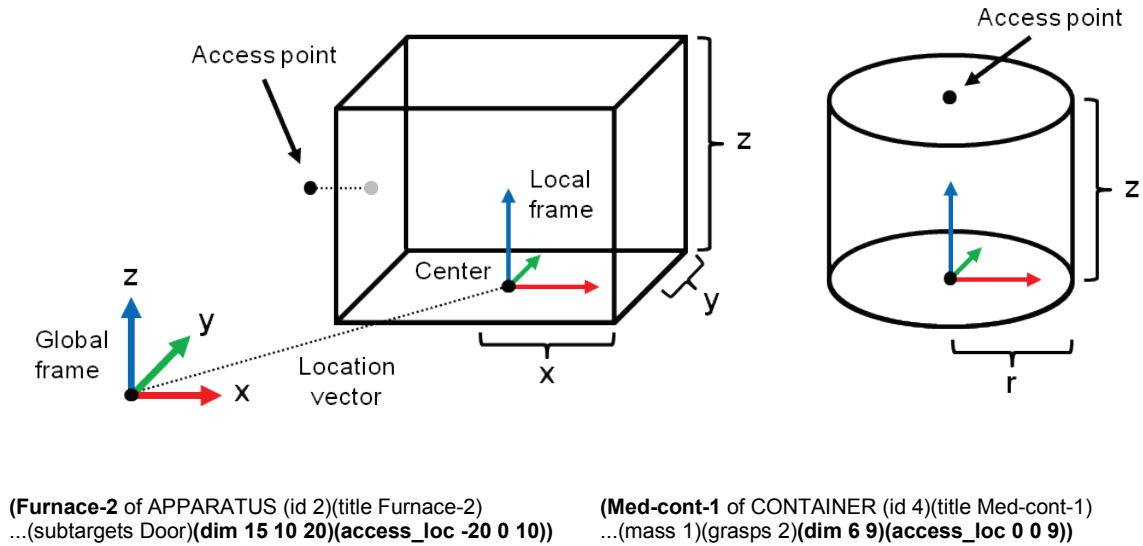


Figure 7.1: Representations of component geometry for box and cylindrical shapes with example component instances below.

The actual value stored in the *dim* slot is half the full x and y dimensions (or full radius for cylinders) and the full z dimension. The component bounds are found by moving the corresponding dimensional distance outward from the local origin in the direction of the positive and negative x and y axes and positive z axis. This formulation

was selected to better suit collision modeling, which is described in Section 7.3.2.1. The access point (stored in the *access_loc* slot) for cylindrical containers or other components that can be accessed from any direction is the point on the local z-axis at the component height (Figure 7.1). For other components, the access point is defined relative to the local frame and is typically outside the component volume along one of the x or y axes. A two-dimensional representation of geometry is seen in Figure 7.2. This representation is used often in the configuration process to simplify and speed up calculations.

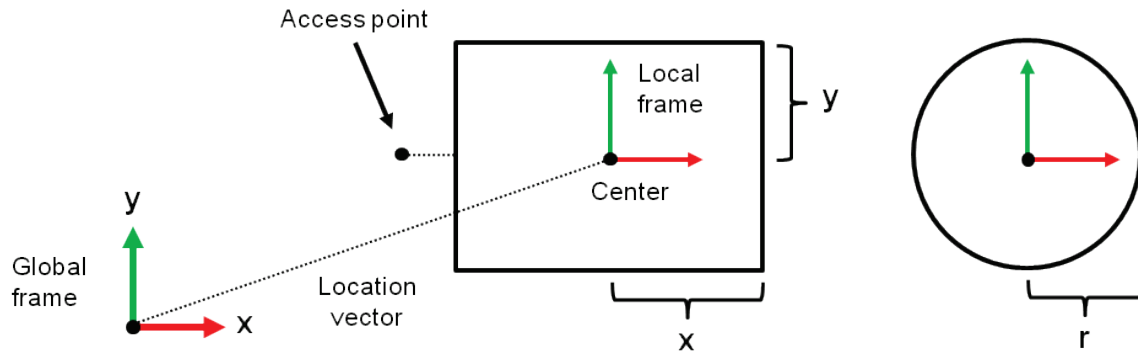


Figure 7.2: A 2-D projection of 3-D component geometry on the glovebox floor (x-y plane).

Many layout techniques also approximate machines to general shapes even if more detailed information is known [Tay, 1996][Mata, 1998][Barral, 2001], so simplifications similar to ours are common. However, our database components are generic, so geometric approximations need to be close to typical component dimensions to ensure that substitution with a specific component instance is valid or an additional layout feasibility check may be required after optimization.

7.1.2 Design Variables

Each component has a *location* multislot and *theta* slot for storing component position and orientation, respectively. The *location* slot stores every location that a component visits during the process as x, y, and z triples. Thus, a flow's *location* slot is added to whenever it is moved, and the initial flow location is the first triple in the first slot, the second location is the second triple, and so on.

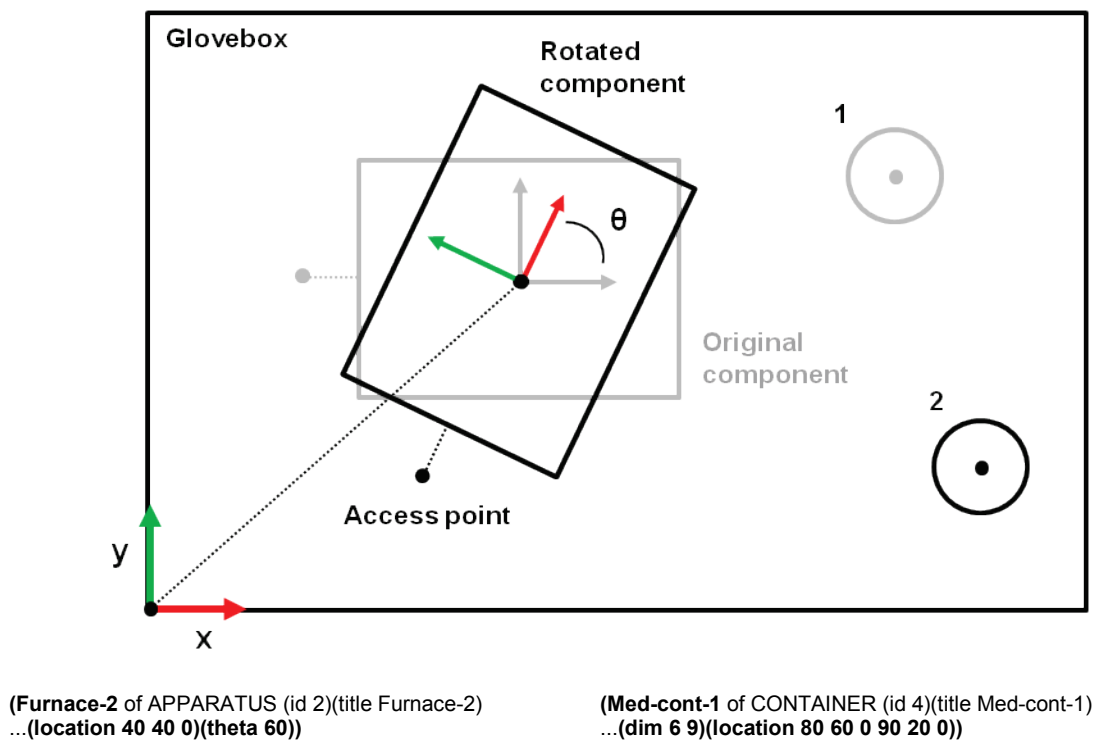


Figure 7.3: A box component rotated by θ and two container locations (1 & 2) with CLIPS instance representations.

The *theta* slot represents a flow's rotation about the vertical axis running through its origin/center. Only a single orientation angle is considered since, in glovebox work, most components (containers, apparatuses, etc) sit flat on the glovebox floor. Since

access points and dimensions are always defined relative to the component's origin in the instance description, calculations must be performed to find the access point and corner points relative to the glovebox origin. The human operator's location is also variable, although a *theta* value is not used. Valid human locations are at the selected glovebox gloveport locations (and represent an additional constraint). Instead of populating multiple positions in the *location* slot, the human's location for a *Move* is tracked using the *origin* slots of the human *Moves*.

Given this representation, our design variables are stored in each component's *location* and *theta* slots. The location variables can be integers or continuous depending on the chosen pattern. A common criterion for terminating pattern search is when the step size reaches a certain value. This value is set to one centimeter in most of our layouts. The smallest unit for rotations is one degree. Differences in component location and orientation that are this small will not show significant differences in layout performance.

7.1.3 Configuration Information

Workcell configuration information is stored in a design's task plan. The plan provides the sequence of operations and material flow which is important for updating flow locations and ensuring proper connectivity. Specifically, each *Move* function represents a change in component and/or material location and the order that components are manipulated by operators. The *Move* functions contain *from* and *to* slots to describe where a flow is being moved from and where a flow is being moved to, respectively. When locating components, the *from* and *to* components in a *Move* must be reachable by the operator listed in the *component* slot from the *origin* defined in the *Move*. Also, a *Move*'s *materials* slot stores the *initial* location of all radioactive materials in the

glovebox before the *Move* is executed. When a material is inside another component, its location is set as the current location of the storing component. The information in the *materials* slot is utilized during the dose calculation (Section 7.2.2.2).

7.2 OBJECTIVES

The layout optimization focuses on material handling. Objective calculations are made each time a *Move* is performed. Our objective function includes the dose received by all operators during a *Move* and the time it takes to complete the *Move*. There are notable differences when evaluating human and robot labor for these two objectives. In particular, reducing the dose received by humans is of utmost importance and the task time calculation can be significantly different based on the difficulty of manipulation for a given operator and the operational constraints (ex: velocity limits for robotics). Such preference information and differences in manipulation abilities will be incorporated into the cost function.

7.2.1 Cycle Time Minimization

While dose minimization is very important, we also need a metric to assess the efficiency of the workcell layout. This is typically given as the total cycle time or distance traveled for a process [Tay, 1996][Mata, 1998][Pashkevich, 1998][Barral, 2001]. The time calculation is derived from the task trajectories. Instead of using distance traveled, we calculate the cycle time to incorporate the differences in velocity between human and robot movements. The robot is also dependent on motion initiation from the control system based on sensor feedback, human input, etc, whereas the human can likely initiate tasks and move on to subsequent tasks much more quickly. We assume a constant

operator velocity for all moves. With a manipulator model, we would have velocity and acceleration limits and could define a trapezoidal velocity profile. In general, accelerations and decelerations occur in a short time-frame for both humans and robotics, so the assumption of constant velocity is generally valid.

7.2.1.1 Move Tasks

The total cycle time is derived from a concept's *Move* functions once the layout is finished. The distance is calculated for the movement of the human's hand or robot's end-effector (EEF) which is tracked during the entire process. Given a *Move* function, two waypoints are established based on the current component locations in the *from* and *to* slots, respectively. Based on these components, the total distance is the sum of three moves:

- An approach move from the current hand/EEF position to the access point of the *from* component,
- A move between the access points of the *from* and *to* components, and
- A retract move from the *to* component's access point to a defined "clear" location.

These are all linear distances. The presence of obstacles, which leads to the need for curved trajectories, influences penalty calculations (Section 7.3.3). The retract move is to a location a fixed distance above the *to* component and along the vector from the *to* component to the operator origin. The final retract location becomes the new current hand/EEF location for the approach move for that operator's next *Move*.

We also assume fixed manipulation times for opening and closing doors and locks. In these cases, the *from* and *to* components are the same, so the typical intermediate move between these components becomes the manipulation time. This is an

approximation based on experience with typical sub-target types and is large enough to produce a noticeable difference between operation of components with and without sub-targets. In most cases, this time estimate should be lower for a human, so we can multiply the human manipulation time by a pre-defined factor greater than one to get an approximation for robotic manipulation.

7.2.1.2 Other Tasks

The robot location is constant during the whole process, but the human's location changes when moving between *gloveport sets/pairs* (one for each arm) for adjacent *Moves*. This is necessary if all the system components cannot fit within the workspace of a single gloveport pair. In these cases, a time is estimated for the gloveport change. This involves the time to:

- Move a hand from its last location back to the “home gloveport location”,
- Remove hands from gloves and perform a radiation survey,
- Walk to the next gloveport pair, and
- Enter the new gloves.

The starting point for the approach move becomes the “home gloveport location” of the new port. The rest of the move is calculated as normal. Much of this gloveport change calculation is based on experience, but the approximation is of a reasonable order of magnitude that clearly adds a “penalty” for glove changes. Recall, that the human operator origin/location for a *Move* is a variable.

Differences in the time to perform a non-material handling function will also exist between concepts that utilize manual labor rather than an apparatus (ex: breaking by hand vs. breaking with a press). However, for the purpose of *evaluating a layout for a single*

concept, these manipulation times are not important. Here, the focus is on material handling for a set task plan and the *location of the manipulation areas* relative to each other. The time to break a material is relevant when different designs (with different function embodiments) are being compared. At this point, the process expert would approximate manual operation time and apparatus time and add these estimates to the total material handling time found from optimization.

Thus, many manipulation approximations can be built into the model based on the necessary completeness of the calculation. Some time estimates are essentially constants that are added to all layouts and do not influence solution convergence, only the value of the final time calculation. Other estimates influence convergence or design comparisons by adding “penalties” for certain variable or component selections.

7.2.2 Dose Minimization

The total operator dose comes from all the nuclear materials that are in the glovebox. The material may contain a single radioactive nuclide or multiple radionuclides. For dose calculations, the dose received from each nuclide present must be incorporated. We also only deal with external dose as internal uptakes only occur in accident or abnormal scenarios and not in normal operations.

The types of radiation typically encountered include alpha, beta, and photons (gammas and x-rays). Gloveboxes are employed to prevent the spread of contamination and because materials are strong alpha emitters. Alpha and beta radiation are usually blocked by the stainless steel and glass of the glovebox and the glove materials. The dose received from photons is a much greater concern. Neutrons are also emitted from materials, but are typically low for most LANL processes, except those working with Pu-

238. The process materials will determine which radiation types are incorporated into the dose calculation.

The dose calculation can take many forms depending on how precise a calculation is needed. Analytical formulas can be derived for a number of scenarios involving different combinations of nuclides, source shapes, attenuation materials, and desired dose metrics. These approaches commonly utilize pre-computed data (ex: dose conversion factors, interaction properties, etc) drawn from large databases/tables. Additionally, more sophisticated Monte Carlo techniques (ex: MCNPX) can be utilized when complex material and layout geometries are encountered. Here, we stick to a simple dose calculation formulation that allows for easy computation during the optimization process. A more in-depth discussion of concepts presented below can be found in Shultis [2000] and Cember [2009].

7.2.2.1 Dose Derivation

At a basic level, the dose rate is related to the activity of a nuclide and the energy deposited from interactions with its emitted particles. The activity of a radioactive material is given by

$$A = \lambda N \quad (7.1)$$

where λ is the nuclide's half life and N is the number of atoms. The number of atoms is related to the nuclide mass m by the equation

$$N = \frac{m}{AW} N_A \quad (7.2)$$

where N_A is Avogadro's number and AW is the nuclide's atomic weight.

The unit of activity is the becquerel which represents one disintegration per second or one atom transformed per second. The becquerel is “a measure only of quantity of radioactive material” [Cember, 2009] because one or more particles may be emitted per disintegration/transformation. For instance, a nuclide may emit a single particle each transformation, multiple particles each time, or at least one particle and one or more particles a certain percentage of the transformations.

We can derive the fluence rate or flux (particles per area per unit time) from the activity. For each type of particle p emitted (with energy E_p), the particles produced per second is

$$S_p = Af_p \quad (7.3)$$

where f_p is the fraction of transformations that emit particle p . Assuming a point source emitting particles radially outward and isotropically, the *uncollided flux* a distance r from the source is

$$\phi_p(r) = \frac{Af_p}{4\pi r^2} \quad (7.4)$$

since each unit area on an imaginary spherical shell with radius r has an equal number of particles crossing it [Shultis, 2000]. The total flux would be the sum of the fluxes from all particles emitted by the material.

A human's *response* to ionizing radiation can be considered as his or her dose. The *absorbed dose* is related to the amount of particle energy absorbed per unit mass (ex: photon energy absorbed in human tissue). The SI unit for absorbed dose is the gray (Gy), where one gray equals 1 Joule absorbed per kilogram. The *dose equivalent* takes into account differences in the biological effects produced for different radiation types. This dose contains a quality factor that multiplies the absorbed dose and is one for photons. A higher factor represents a larger response/effect. Furthermore, the *effective dose equivalent* or *effective dose* takes into consideration that certain organs and tissues are more radiosensitive than others utilizing a quality factor for specific tissue/organ types. These two latter doses are typically expressed in rems or sieverts (Sv).

The response R due to an amount of energy ΔE transferred to a medium due to all interactions within a sufficiently small volume ΔV with mass $\rho \Delta V$ can be given by [Shultis, 2000]

$$R \equiv \frac{\Delta E}{\rho \Delta V} = \epsilon \frac{\sigma N}{\rho} \Phi \quad (7.5)$$

where ϵ is the average amount of energy transferred in a single interaction, σ is the microscopic cross section for the interaction, N is the number of atoms *per unit volume*, ρ is the density of material in ΔV , and Φ is the fluence (particles per unit area). This is for a target at a single point in space that responds equally to radiation in all directions and a fluence of monoenergetic particles. A generalization of this formula for photons [Shultis, 2000] is given as

$$R(\mathbf{r}) = \int_0^\infty dE \frac{\mu_d(\mathbf{r}, E)}{\rho(\mathbf{r})} E \Phi(\mathbf{r}, E) \quad (7.6)$$

where μ_d is the *linear disposition coefficient* and is given by

$$\mu_d(\mathbf{r}, E) = \sum_i N_i(\mathbf{r}) \sum_j \sigma_{ji}(E) \frac{\epsilon_{ji}(E)}{E} \quad (7.7)$$

where the subscript i refers to the atomic species or isotopes in the material and the subscript j indicates the specific type of interaction. The *response function* \mathcal{R} relates/converts the radiation fluence Φ to the dosimetric quantity D and can be seen as

$$\mathcal{R}(E) = \frac{\mu_d(E)}{\rho} E \quad (7.8)$$

from Equation 7.6. The mass energy deposition coefficient $\mu_d(E)$ selected depends on the type of response to be calculated (ex: absorbed dose, effective dose, etc). Common units for the response function are $Gy\ cm^2$ or $Sv\ cm^2$.

Response functions have been calculated and tabulated for a number of scenarios. In addition to the radiation type and energy, the response function value is dependent on the phantom used to represent the human (ex: ICRU sphere, slab and spherical, or anthropomorphic) and the incidence of the radiation field (ex: anteroposterior, posteroanterior, lateral, rotational, or isotropic). More in-depth descriptions for response function calculations can be found in [ICRP, 1987][ICRU, 1992][Shultis, 2000].

For *monoenergetic* particles, putting the response function and fluence together gives the uncollided dose

$$D(\mathbf{r}) = \mathcal{R}\Phi(\mathbf{r}) \quad (7.9)$$

whereas using the flux gives the uncollided dose rate for a point source

$$\dot{D}(r) = \mathcal{R}\phi(r) = \mathcal{R} \frac{Af_p}{4\pi r^2} \quad (7.10)$$

If a shield or other material is present between the source and human, then particle attenuation occurs as source particles interact with the material as they travel. The resulting dose rate from *material attenuation* can be given by

$$\dot{D}(r) = \mathcal{R} \frac{Af_p}{4\pi r^2} \exp(-\sum_i \mu_i t_i) \quad (7.11)$$

where the μ_i is the attenuation coefficient of material i with thickness t_i . The $1/r^2$ factor is sometimes referred to as the *geometric attenuation*.

The neutron dose rate can be calculated similarly from the source strength and the neutron response functions. Compared to photon calculations, there are differences in notation and the calculation of energy transferred (ϵ) during interactions.

7.2.2.2 Dose Calculation

For our examples, we will focus on the absorbed dose from photons. The following simplifications and assumptions are used in our calculation:

- Nuclear materials are considered as point sources – Materials are typically grouped in one location, of smaller batch sizes, and at adequate distances from the operator to warrant a point source approximation.

- Assume no attenuation from other materials/shields – Glovebox attenuation is consistent for all designs. Attenuation by components between a radionuclide and the operator is minimal. Shields and/or shielded containers are not components needed to perform process functions, so they are not considered in the scope of our generated concepts.
- No photon buildup is incorporated – Large material mean free paths are not likely to be encountered and component material composition may not be known. However, when lower energy photons are expected, there may be a higher potential for photoelectric interactions with high Z materials, but the effect of these interactions should be minimal.
- Assume a linear trend for dose versus detriment.

Most of these simplifications and assumptions are chosen to simplify calculations for the optimization algorithm and because more detailed material and geometric information is not known.

Most aspects of the dose calculation can be made ahead of time. Based on the material constituents and their masses, the most dominant photons emitted during transformation, the response functions for the photon energies, and the proper conversion factors, we can calculate all the variables in Equation 7.10 except the geometric attenuation,

$$\dot{D}(r) = \mathcal{R}\phi(r) = \frac{\mathcal{R}}{4\pi r^2} \lambda f_p \frac{m}{AW} N_A = \frac{C}{r^2} \quad (7.12)$$

The computed factor C that multiplies $1/r^2$ is stored in a material's *dose* slot. This slot value would contain the sum of contributions from all considered particles emitted by the

material. The calculation of C is nearly identical to an approximated calculation of the *specific gamma-ray constant* Γ but with a specific mass/activity defined.

The dose calculation is made after the layout is constructed. For each *Move* function, the evaluation module retrieves the nuclide locations stored in the *Move*'s *materials* slot, determines the current operator locations, computes the separation distance, and then calculates the dose for each operator from each material. This dose is the value from the *dose* slot (C) divided by the separation distance squared. If a material is being moved, the separation distance varies with time. In this case, we determine the closest distance between the operator and the material's initial and final locations and use the closest distance as the separation distance for the entire move. This speeds up the dose calculation and provides a more conservative dose estimate. In our model, it is assumed that all operators are present during the entire process and receive a dose for every operation performed in the glovebox whether or not they are involved in the task.

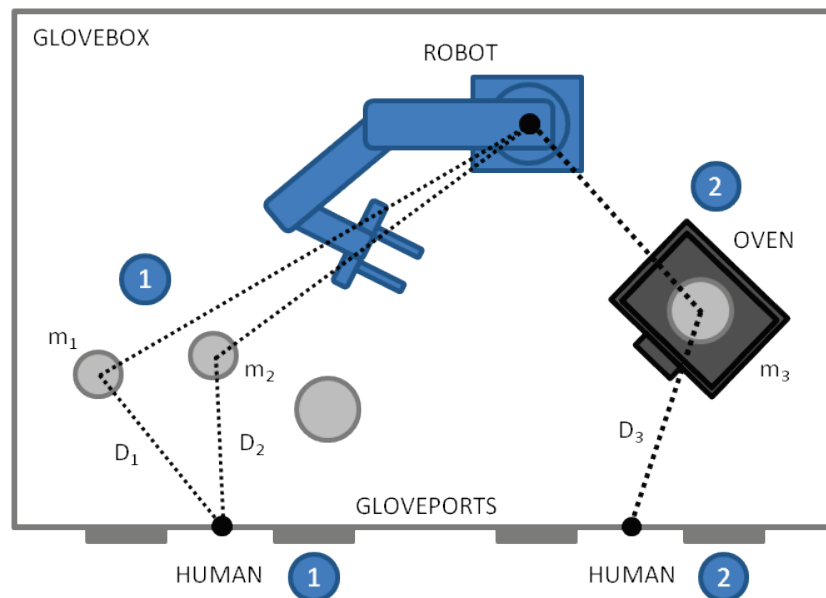


Figure 7.4: Calculating robot and human dose for two different *Moves* in the task plan.

Figure 7.4 shows two different dose calculation scenarios for different stages in the glovebox process. In the first case (1), the materials m_1 and m_2 are in separate containers and separated from the operator by distances D_1 and D_2 , respectively. At this time, the human is working at the leftmost gloveport pair and receives a dose from each material based on its separation distance. The robot dose can be similarly calculated. In the second case (2), materials m_1 and m_2 have been combined into a single container to form material m_3 . The human is now at the rightmost gloveport pair and receives a dose from m_3 at a distance D_3 , which is the combined dose from materials m_1 and m_2 at that same distance.

7.2.2.3 Robotic Dose

The radiation dose effect on the robot is hard to quantify. In most cases, the typical cumulative dose until failure (in rad or Gy) can be estimated for robot electronic sub-components and non-metal materials [Sharp, 1996][Youk, 1999]. Major contributions to the total dose come from photons and neutrons. In plutonium operations, however, the photon and neutron levels are not likely to be a large concern, but the alpha radiation effects can be a problem [Heywood, 1990]. The intense alpha emission of Pu can result in a “dust” that collects throughout the glovebox affecting seals and other non-metal materials. Additional dust will collect in the glovebox when oxides are handled. Dust can also have abrasive effects at mechanical interfaces. There is no easy means of calculating the quantity of “dust” present and its effect.

Thus, there are potential robotic failure effects from numerous radiation types. At this point in design, however, we do not know the desired system lifetime, the typical run cycle, or sub-component specifics to relate dose and sub-component effects/failures in a

meaningful way. A full analysis of radiation effects on electromechanical components and robotic failure analysis is beyond the scope of this work.

However, although vague, similar to a human, there is some known detriment associated with each dose that a robot receives. Radiation can degrade component parameters over time and decrease the component lifetime from the typical lifetime under non-radiological conditions [Lauridsen, 1996]. Thus, in theory, lowered dose should lead to a smaller reduction in the lifetime or mean time between failures and decrease the frequency of maintenance – a general benefit to the whole robotic system.

One approach to incorporate “robotic dose” is to calculate the exposure or absorbed dose for the robot relative to its location, which is reflected in the calculation of the factor C . In this way, we get an estimate for robotic detriment per process run. We then normalize this dose and add it to the objective function multiplied by an appropriate weight to reflect the preference of robot dose to human dose.

Actual estimation of robot influence on workcell performance/evaluation is complex. This analysis requires detailed technical and process information and a failure modes and effect analysis. Some parts may require replacement or maintenance quite frequently to avoid failure. Expected system maintenance leads to maintenance worker dose and down time that must be factored into the ultimate cost of the system. At a rough level, the robot dose will serve as a factor to leverage these future costs into the performance of a design concept.

7.3 CONSTRAINTS

The following inequality constraints are utilized in our formulation:

- Components must be located inside the designated glovebox

- No components can overlap/collide
- Collision-free reachability of component access points

There are no analytical functions of the design variables that can represent these constraints. Instead, CLIPS computational sub-functions perform these three constraint checks on each proposed new component location. In regard to these constraints, our model is formulated as an unconstrained optimization problem. The following sections discuss how constraints are checked and the formulation of penalty functions. The constraint that the human must work from one of the glovebox gloveports is always enforced/propagated. Thus, a feasible human location/gloveport is selected for each *Move* and no penalty is associated with this constraint.

7.3.1 Within Container Bounds

CLPIS performs a check to see if any part of the component extends beyond the glovebox bounds when a new location is proposed. If a violation occurs, the boundary penalty B_{id} for component i in dimension d is

$$B_{id} = \frac{|p_{id} - g_d|}{|p_{id} - c_{id}|} \quad (7.13)$$

where g_d is the value at the violated glovebox boundary in dimension d , p_{id} is the value of the furthest point of component i from the boundary in dimension d , and c_{id} is the component's center value in dimension d . For example, if the center of a cylinder with radius r is placed a distance $2r$ beyond the lower boundary, then the furthest point is a distance $3r$ from the boundary for a penalty of $3r/r$ or 3 (Figure 7.5). This penalty is essentially the factor of the component's "width" from its center to edge that must be

translated in dimension d to remove the violation. The total penalty P_B is a unit-less quantity and is the sum of all boundary violations in the x and y directions for all components. A buffer distance could also be defined so a penalty is assessed if the component is within the bounds but less than a minimum distance from the wall.

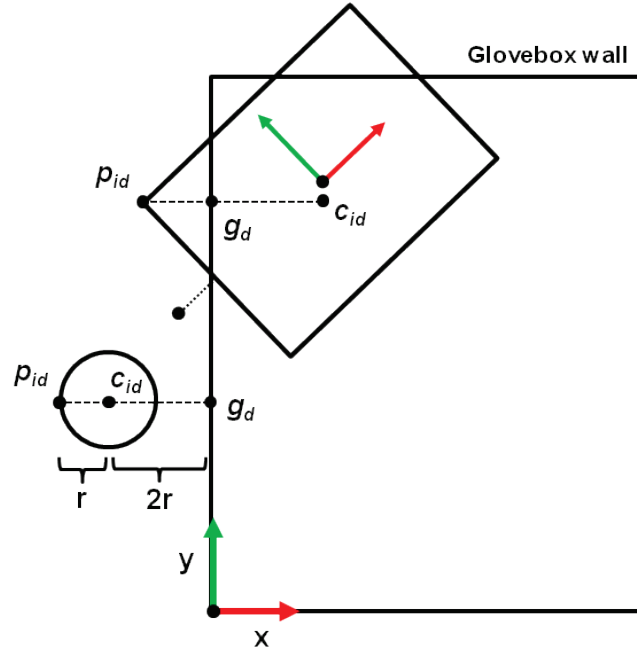


Figure 7.5: Calculating the penalty for component boundary violations.

7.3.2 No Component Collisions

In our problem, no proximity or adjacency constraints are placed on the design. Cycle time minimization will help drive components used in the same *Move* closer together. The only component-component condition is that they do not overlap/collide. Components are either cylindrical or box shaped as defined by their *dim* slots, and all component features are assumed to be contained within this volume – no odd-shapes are

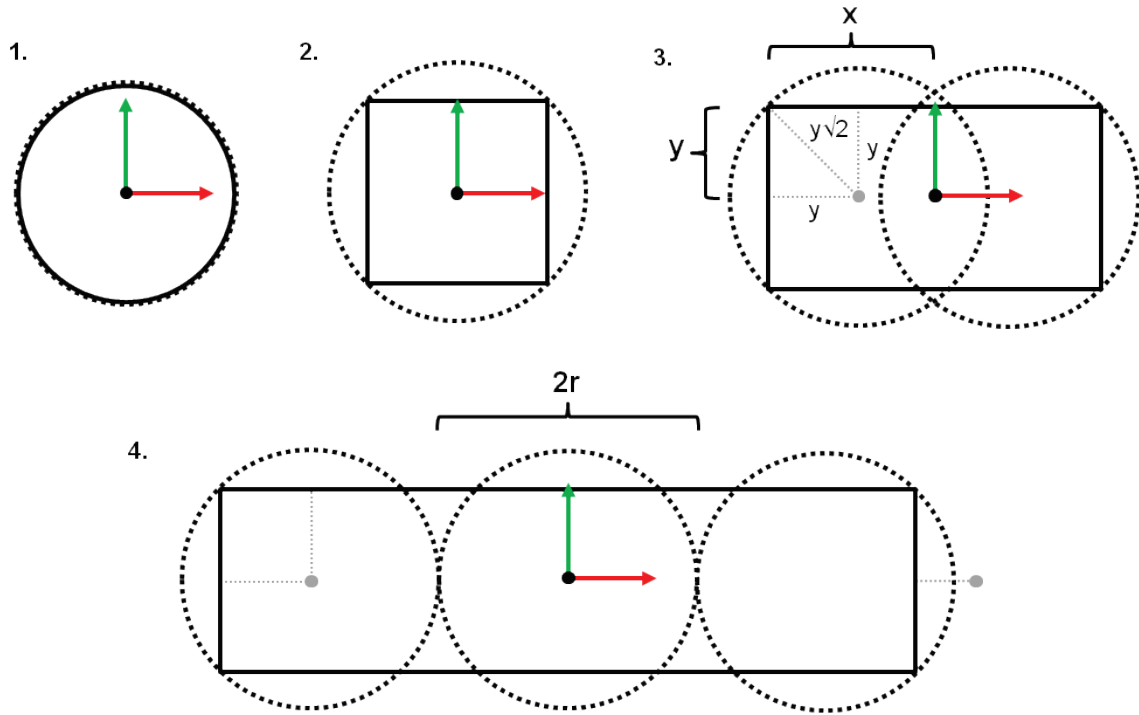
considered. Depending on the number n of already placed components, checking a single component for a single new location could involve at least n component-component collision checks and more if components have multiple locations to check. Thus, each component has a simple collision model for quick collision calculations.

7.3.2.1 Collision Model

The collision model transforms each shape into one or more cylinders that approximate the component volume (Figure 7.6). Each cylinder is defined by a circle and the component height. The method for defining the number and size of *circles* is similar to that of Mata [1998]. Each circle has the same radius and a center location defined relative to the component's local frame. The circle information is stored in the component's *circles* slot, where the first value is the radius and the following triples are the circles' centers.

A cylindrical component has a single circle with a radius equal to the radius in the *dim* slot and a center that coincides with the local origin "(0, 0, 0)", as seen in (1) of Figure 7.6. In the case of a square (2), the four square corners all intersect with a single circle to completely enclose the area. For general rectangles (3), the circle radii are equal to the minimum of the x and y values in the *dim* slot multiplied by $\sqrt{2}$ so that circles placed on each of the long ends intersect adjacent corners. The center locations are appropriately calculated and added to the *circles* slot. Then, if the space between the centers of the two end circles is greater than twice the radius, additional circles are added. The number of circles added is equal to the separation distance between end circle centers divided by twice the radius, rounded down. The centers of the additional circles are

placed on the line between the endpoint circle centers and are spaced so that the distance between any two adjacent circle centers is constant.



(Furnace-2 of APPARATUS (id 2)(title Furnace-2)...(dim 22.142 10 20)(circles 14.142 0 12.142 0 0 -12.142 0))

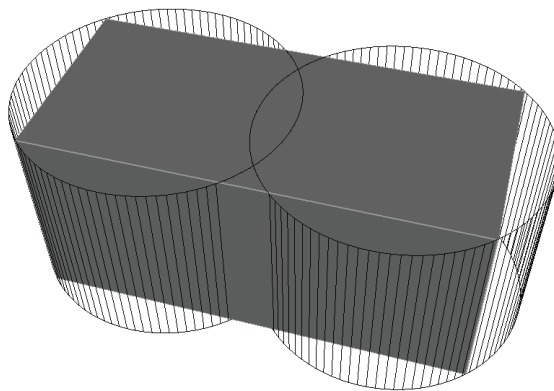


Figure 7.6: The simple component collision models from a 2-D perspective (1 – 4) and an instance and 3-D representation for a component similar to (3).

In Figure 7.6, example (4) represents a “worst case” in coverage, as a slightly longer length would add another circle, increasing coverage. At the worst case, for a very large number of circles, about 9.1% of the actual component area is uncovered. In many cases, only one to three circles will be used, with missed coverage likely lower than 5% and in many cases full-coverage as in (1) – (3). The overestimated cylinder footprints provide a buffer between components. If desired, an additional buffer could be added.

7.3.2.2 Collision Calculation

Since components can only rotate about their local z-axis and are upright, we only need two checks to find a collision (Figure 7.7):

- Whether the 2-D (x-y) cylinder footprints (i.e. circles) of two components intersect – This calculation is very quick, requiring that the separation distance of the circle centers be greater than the sum of the radii to avoid collision.
- If there is a footprint/circle intersection, check if one component is above and entirely clear of the component below it – This calculation requires that the separation distance between the component bases be greater than the height of the lower component to avoid collision.

The collision check is performed between the newly placed component and every other component that is already placed.

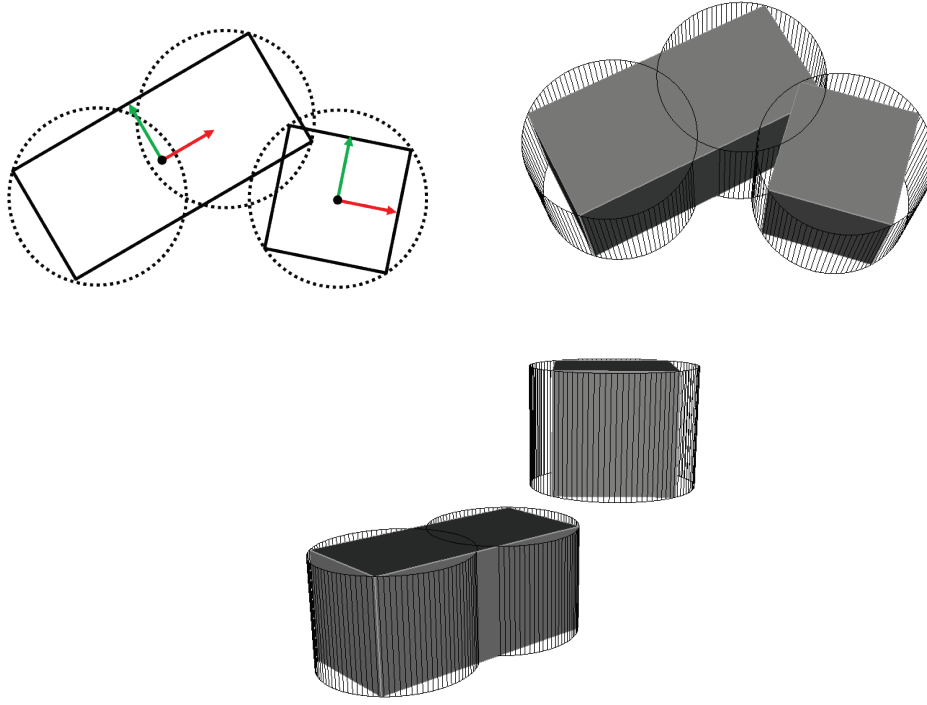


Figure 7.7: A collision between two components (top) and the same 2-D projection circle intersection but with no 3-D collision (bottom).

The penalty calculation for collisions is based on circle overlaps of different components. For two circles c_j and c_k with radii r_j and r_k and locations l_j and l_k , respectively, the overlap penalty C_{jk} is

$$C_{jk} = \frac{(r_j + r_k) - (mag_{2D} l_j l_k)}{(r_j + r_k)} \quad (7.14)$$

where mag_{2D} finds the distance between l_j and l_k in the x-y plane. No penalty is assessed if C_{jk} is zero or less. The penalty gives the fraction of the radii sum that one circle must translate to remove the violation. The total collision penalty P_C is the following sum:

```

For all pairs of different components  $F_a$  and  $F_b$ 
  For each location  $l_h$  of component  $F_a$ 
    For each circle  $c_j$  of  $F_a$  at  $l_h$ 
      For each location  $l_i$  of component  $F_b$ 
        For each circle  $c_k$  of  $F_b$  at  $l_i$ 
          Compute  $C_{jk}$ 
          Add  $C_{jk}$  to the total penalty sum  $P_C$ 

```

The resulting penalty P_C is unit-less. The penalty for overlap in the z-dimension can be similarly calculated, if desired.

7.3.3 Target Reachability

A target is reachable if its access location is within the operator's reach and a collision-free trajectory exists to the target. To quantitatively define reach, we need rough models of the human and manipulator arms. For a human, we can approximate the shoulder to contain three rotational degrees of freedom (DOF) with the elbow having a single rotational DOF [Lenarčič, 1994][Yang, 2009]. The trunk of the body provides three more DOF [Jung, 2010] that can help place the location of the shoulder joint.

The human's center (Figure 7.8) can be defined as the center-point on a line between the two shoulder joints (i.e. a short distance below the base of the neck). In our designs, we consider serial manipulator arms which have 6 or 7 DOF. The robot center is defined at the intersection of the first two rotational joint axes (Figure 7.8). Based on these models, an operator's reach is the maximum distance between its center and wrist point. These models are very rough approximations but sufficient for our needs.

A path to a component's access location can be generated in either joint space or EEf (Cartesian) space. This trajectory begins with the initial arm configuration before the move and ends with the arm configuration needed to reach the target point. In most

transport tasks, we want the transported object to remain upright along the entire path. Thus, we are interested in Cartesian space planning since we need to control the position and orientation of the EEF. However, for a given hand pose, the joint configuration is still important for determining whether the arm collides with obstacles (i.e. other components).

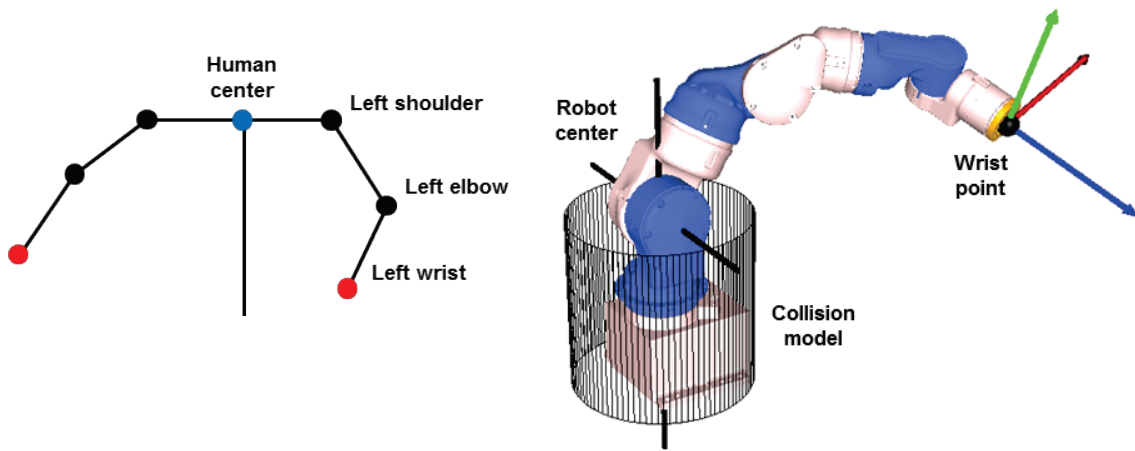


Figure 7.8: The locations of the human (left) and robot (right) centers and wrist points.

7.3.3.1 Path Planning

A common Cartesian path planning approach is to build a trajectory by defining a number of via points that describe the position and orientation of the hand/EEF at various points between the starting arm location and the endpoint (Figure 7.9). Then a smooth path is built through or near adjacent points. Constructs such as Bezier curves and B-splines can be utilized for this task. Cartesian path planning is particularly useful in the presence of obstacles/components when an arbitrarily generated path is more likely to lead to collisions.

Even if an operator can reach all points on a generated path, collisions may still occur depending on the arm configuration needed to attain the desired EEF pose. For manipulators, this collision check typically requires EEF trajectory generation and then simulation of the manipulator moving its EEF along this path to confirm it is collision-free and also that no singularities or joint limits are encountered during the move. Such a simulation is too computationally expensive to perform within an optimization loop.

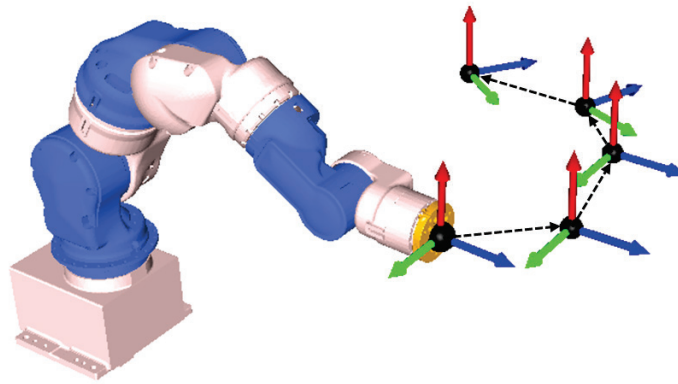


Figure 7.9: An EEF trajectory as a series of via points containing orientation information.

In our layouts, if a path can be found to the target point, that path is most likely collision-free. A valid path requires enough space between components such that the arm can fit through. Based on the reach penalty calculation (to be described), a buffer distance typically exists between components. Also, components sit upright on the glovebox floor with no protrusions or overhangs, so the vertical space between them is clear. We also assume that some empty space exists between the operator and any placed component so that the operator has room to freely move close to its base/origin in preparation for a reach. In general, our results support this collision-free path assumption (Chapter 9).

If a straight path exists between operator and target (i.e. no obstacles between them), the operator should have space to configure its arm such that it can move to the target in the plane containing this path. The target is reachable if the straight-line distance is less than the operator reach. If there is an object between the operator and target, the operator can try to reach around or over the obstacle (Figure 7.10). This results in a final arm configuration that must bend around or over the obstacle which effectively reduces the linear operator reach. We can generate a path that resembles this configuration (Section 7.3.3.3). In this case, we assume the path approximates the operator arm and therefore the path length must be less than the operator's reach to be valid. Given the kinematic arm models and their typical number of links and DOF, this is a reasonable approximation.

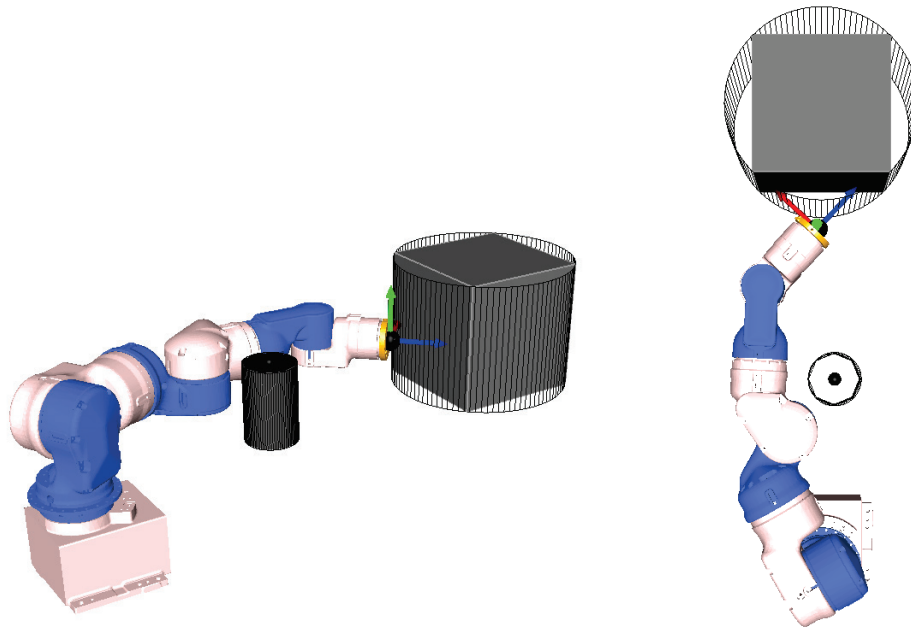


Figure 7.10: A serial manipulator reaching around an obstacle to locate its wrist point at a component's access point.

The path planning we perform is not true EEF path planning but rather calculations that help determine or establish reachability. The “path” or point sequence is not the actual trajectory that the hand should traverse and hand orientations are not considered. The path is used as an approximation of arm configuration to determine operator reach. However, if a generated path directly to a target or moving around an obstacle is less than the operator’s reach and also collision-free, this indicates that there exists a volume between the operator and target that the operator should be able to move through without any collisions. An actual EEF trajectory can then be established in this space.

Based on our discussion, by establishing that a path exists to a target, we can assume that this path is collision free. We can do this for the access points of the *from* and *to* components for a *Move*. We do not check for a collision-free path between these access points and instead assume that if both points are reachable then a collision-free trajectory exists between them. We see this is generally valid in our results and that in many cases a straight-line path is possible between the *from* and *to* components.

7.3.3.2 Obstacle Check

Obstacles between an operator and a target are detected using a component’s collision model (Figure 7.11). The line between the operator’s location and the access point is projected onto the x-y plane (i.e. glovebox floor). Then each circle of each component is checked to see if it intersects the line segment. An intersection occurs if the distance between the circle center and the closest point on the line segment to the circle is less than the circle’s radius (or the radius plus some buffer distance).

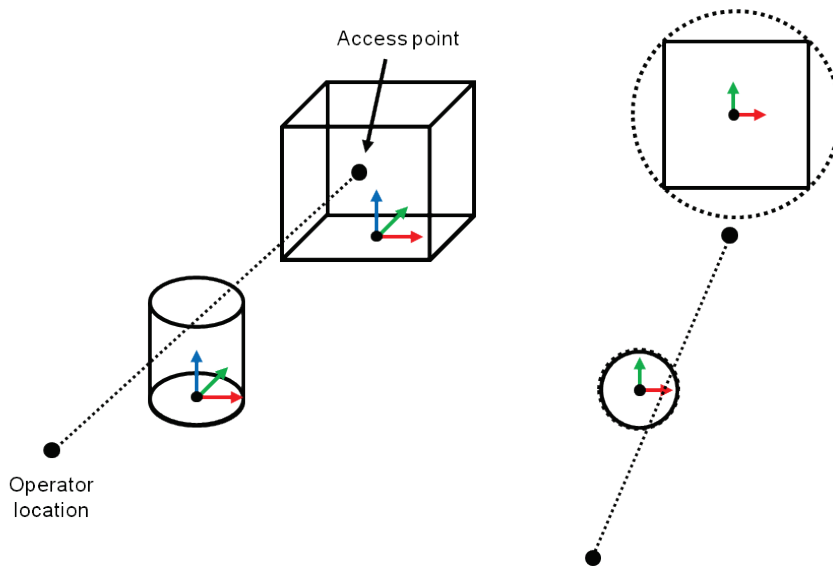


Figure 7.11: Detecting an obstacle between the operator and access point in 3-D (left) and 2-D (right).

7.3.3.3 Path Generation

We approximate a path as a series of via points connected by linear segments. These paths are typically limited to one to three links so they can also approximate an arm configuration when reaching around obstacles. The three path directions are heuristic in nature and assume that the obstructing object cannot be moved for that reach. When an object is in the way, the typical human response is to reach around the object on either side or reach over the object. These are the three paths we generate, and they are developed using a consistent set of operations. There is no search for the optimal path which would be too costly. Instead, we find the shortest of the three generated paths and compare it with the operator's reach to determine reachability.

Similar to obstacle checking, two-dimensional projections (i.e. line segments and circles in the x-y planes) are used to develop the x and y positions of via points around

obstacles. The z-values of these points are filled in separately. The paths are calculated differently depending on how many circles exist in the collision model.

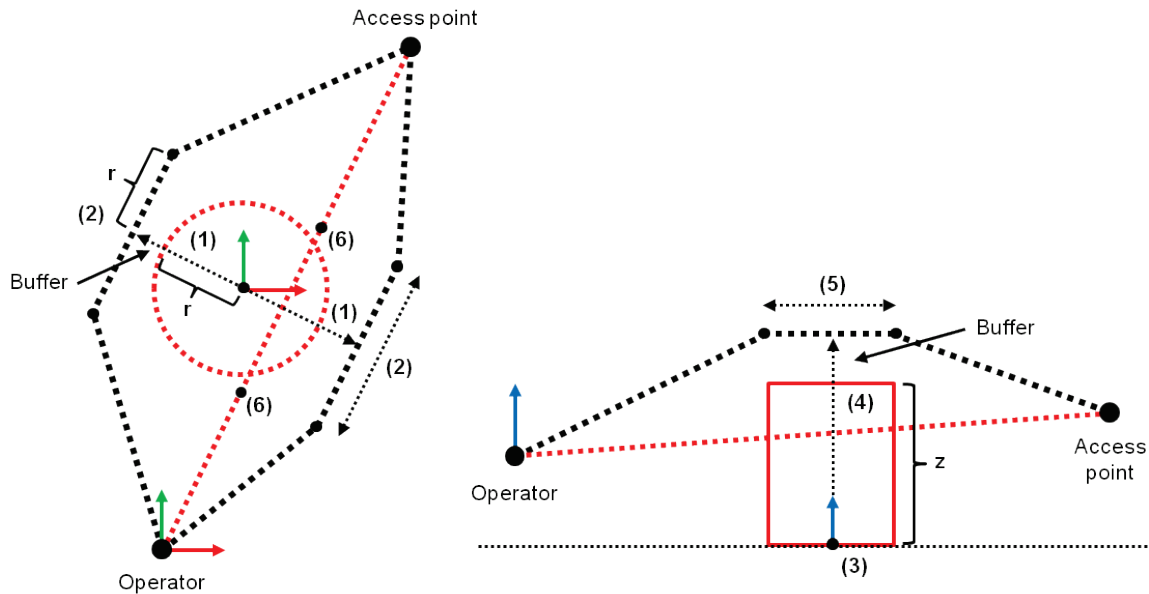


Figure 7.12: The path formulations for going around (left) and above (right) an obstacle with a single circle.

When a single circle is present (Figure 7.12), for the side paths, moves are made outward from the obstacle center and perpendicular to the line segment a distance equal to the radius plus a buffer distance (1). From this point, two way points are established heading in opposite directions a distance equal to the radius along a line parallel to the line segment (2). The z-value for these points is set to the average of the operator and access point heights. Both paths contain two intermediate via points to maneuver the hand around the obstacle.

A similar technique is used to find the path over an obstacle which intersects the vertical plane containing the end-points (Figure 7.12). For a single circle/cylinder, the

closest point to the circle in the x-y plane on the line segment between the endpoints is first found (3). The z-value of this point is set to the sum of the obstacle's z-position and height plus a buffer distance to construct a point higher than the obstacle (4). Two other points are placed on either side by a distance equal to the obstacle's radius (5). These two points also reside in the vertical plane containing the end-points (6) and represent two intermediate via points for the over trajectory. One or both intermediate via points may be removed depending on whether one or both endpoints are higher than their respective adjacent points. Three-dimensional representations of these paths are seen in Figure 7.13.

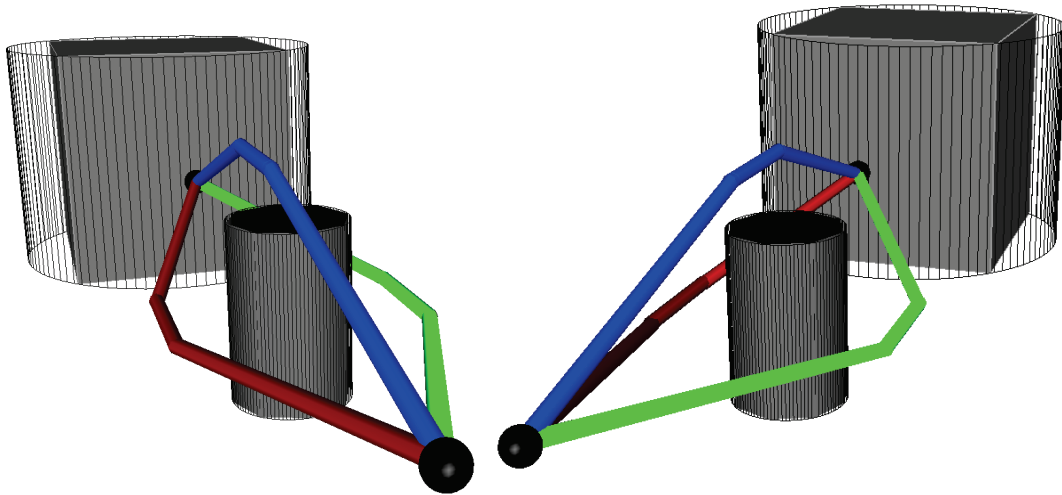


Figure 7.13: Two views of example 3-D paths for going over and around an obstacle that has a single circle.

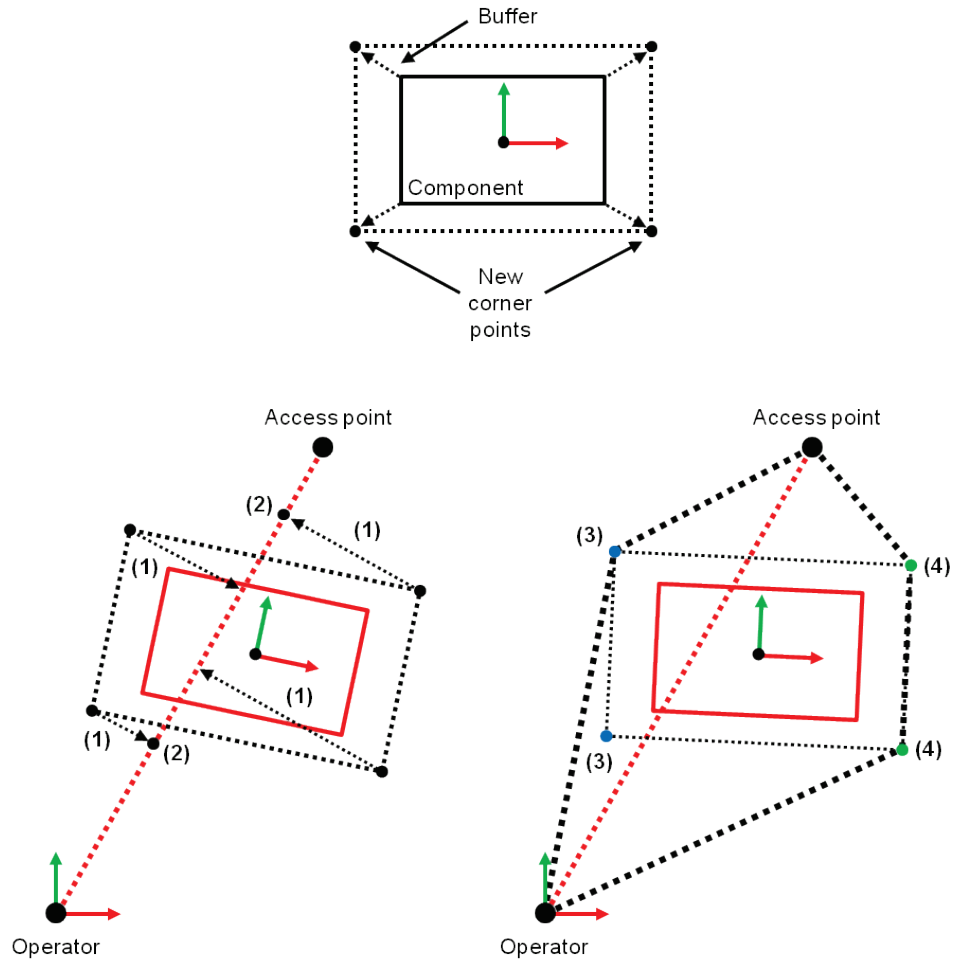


Figure 7.14: Extending an obstacle's corners in the 2-D plane (top) and determining the intermediate points for paths over (left) and around the obstacle (right).

When more than two circles are present, the path generation is different. To find a path over a box-shaped obstacle (multiple circles), the collision model is not used. We first find the corners of the obstacle's 2-D projection on the x-y plane. We move these points outward along a vector from the component center to the original corners to create a new rectangle which now includes a buffer region (Figure 7.14). These points are projected onto the line segment (1), and we find the closest point to the operator and to

the endpoint, respectively, to establish two via points (2). The z-value of these points is set using the same calculation for cylindrical obstacles. The result is two intermediate waypoints higher than the obstacle. This over path model for box-shaped components is not as compatible with components that are both very long and thin, which are not commonly encountered in our designs.

For the side paths, we again exaggerate the four obstacle corners of the 2-D projection (Figure 7.14). The line segment between the operator origin and target point separates the corner points to those on the left side (3) and right side (4). For a single side, the points on the convex hull of a polygon containing the component corner points and the endpoints become the via points for a path going around the obstacle on that side. The z-value is set to the average height of the endpoints.

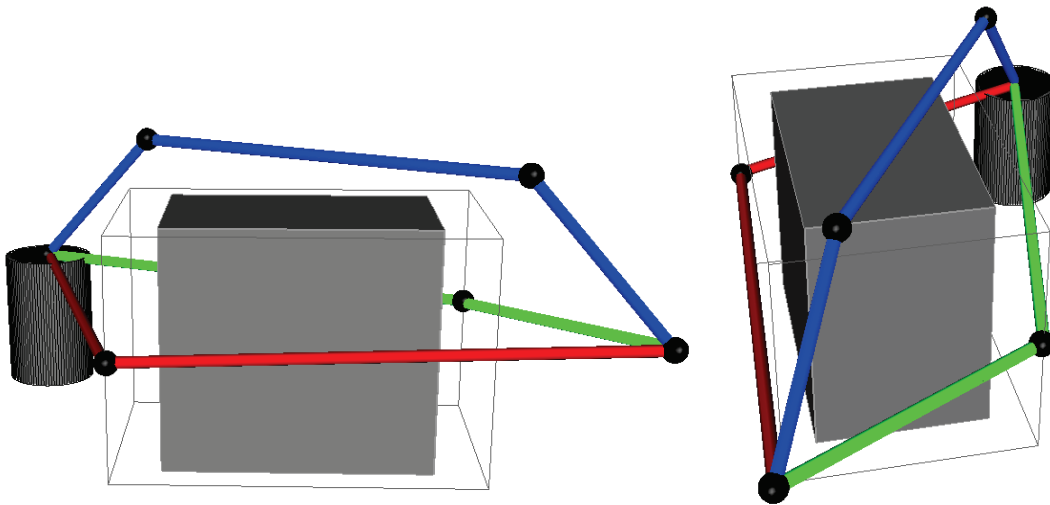


Figure 7.15: Two views of example 3-D paths for going over and around an obstacle that has multiple circles.

A path is invalid if the sum of its linear segments is greater than the operator's reach. For violations, the reach penalty RP_{mip} for operator O_m with reach D_m reaching around an obstacle to access component F_i is

$$RP_{mip} = \frac{(\sum_n s_{np}) - D_m}{D_m} \quad (7.15)$$

where s_{np} is the linear length of the n th segment in path p . The penalty is calculated for each of the three paths. This penalty is the fraction of the reach that the path exceeds the reach.

Each generated path p is also checked for collisions with other components. This penalty is added to the one described above. A similar technique as described in Section 7.3.3.2 is utilized to detect collisions. Each path p contains three segments and collisions with each segment are recorded. In this case, the collision is either a segment passing through a component's circle or an intermediate via point that is inside a circle. For a path p to component F_i , the collision penalty CP_{ijknp} for circle c_j of component F_k with radius r_k colliding with segment s_{np} is

$$CP_{ijknp} = \frac{r_k - \min\left((mag_{2D} \ l_j, l_{cp}), (mag_{2D} \ l_j, l_{ep})\right)}{r_k} \quad (7.16)$$

where l_{cp} is the closest point on s_{np} to the center of circle c_j and l_{ep} is the second endpoint of s_{np} . This check is made for each segment n of each path p for every possible colliding component k . There is no penalty if the path essentially goes above a component.

A reach is always made to a component's access point. If the access point is located on the component (ex: cylindrical containers), then the path segment containing

that endpoint is not checked for collisions with that component. If an access point is defined outside a component's collision model (ex: boxes), a segment-component collision check is made. This ensures that the operator is not “reaching through” the component.

The reach penalty RP_{mip} for a given path p is added to its collision penalty CP_{ijknp} to get the total penalty R_p for the path. The selected path for a given operator-component reach pair is the one with the minimum penalty. These minimum penalties are added for every operator-component reach in the task plan to get a penalty P_R for the entire design.

All the penalties are unit-less by definition. To add the penalties to the objective function, we use a simple penalty function instance of Equation 6.3 where $r = 1$ and $n = 1$. That is, the calculated penalty is added directly to the objective function as

$$P_T = W_B P_B + W_C P_C + W_R P_R \quad (7.17)$$

where the weighting factor W for each penalty is determined when the penalized objective function is constructed (see Equation 8.1).

7.4 SUMMARY

This chapter described our layout optimization model. The problem is formulated as an unconstrained optimization problem. The main objectives are minimization of cycle time and operator dose. Constraint violations associated with boundary violations, component-component collisions, operator-component reachability, and path-component collisions are added as penalties to the objective function. Both the evaluation functions and collision models are formulated for quick calculation within the optimization loop.

These contain a number of simplifications and assumptions, but these conditions are also typical of the conceptual design stage when concrete details are lacking. Thus, our formulation contains a number of arbitrary “buffer” values, but vagueness is not as much of an issue when using pattern search. Furthermore, if desired, the model generality enables the implementation of more detailed knowledge and a number of extensions (some of which will be described in Chapter 10).

Chapter 8: Extended Pattern Search Algorithm

The extended pattern search (EPS) algorithm is versatile and interfaces well with the model presented in the previous chapter. Our extensions are influenced by heuristic knowledge concerning glovebox work. Additional extensions can be incorporated as desired, but the following implemented algorithm efficiently produces useful solutions.

8.1 GENERAL EPS FRAMEWORK

The general structure of an EPS algorithm [Yin, 2000] is presented below:

- Generate an initial layout L_0
- Evaluate $f(L_0)$
- Set the initial translation and rotation step sizes and patterns
- While the stopping criteria are not met:
 - For each component that can be translated
 - Explore a new location using a pattern and generate layout L_1
 - If $f(L_1) < f(L_0)$, accept L_1 as the current configuration ($L_0 = L_1$)
 - Else revert back to the previous L_0
 - If no moves are accepted in the above for loop
 - Reduce the translation step size
 - If step jumping is allowed and the criteria are met, adjust step size
 - For each component that can be rotated
 - Explore a new rotation using a pattern and generate layout L_1
 - If $f(L_1) < f(L_0)$, accept L_1 as the current configuration ($L_0 = L_1$)
 - Else revert back to the previous L_0

- If no moves are accepted in the above loop
 - Reduce the rotation step size
 - If step jumping is allowed and the criteria are met, adjust step size
- If no translation or rotation moves have been accepted and swaps are allowed
 - For a component pair, swap their locations to generate layout L_I
 - If $f(L_I) < f(L_0)$, accept L_I as the current configuration ($L_0 = L_I$)
 - Else revert back to the previous L_0
- End when stopping criteria met

The stopping criteria are typically related to conditions on the step size. The translation, rotation, and swap loops can be performed in any desirable order. Extensions can be made to this general framework to suit the problem needs.

8.2 EXTENSIONS

The exploratory move types described above will be useful in our problem. The use of a pattern for translational moves is particularly beneficial. Instead of randomly trying new locations (as in SA), the pattern allows a “smart” search for new locations about the existing area of the current location using direct search. This is essentially equivalent to utilizing a heuristic – the next best moves are probably those around the existing location that improve boundary violations, collisions, and reachability. Since an evaluation is performed at each move, the algorithm also provides quantitative information about how each move affects the layout objectives.

There are two important factors from glovebox work that influence how and where the move sets/loops should occur in our algorithm:

- *Human operations are constrained to the gloveport locations* – The initial design must have gloveport assignments for human moves. Based on reachability penalty minimization, translational moves should drive components toward the current gloveport assignments. However, these may not be the best option. If other gloveport assignments are tried too late in the configuration process, they have a lower probability of acceptance because enough iterations have passed that the current component locations are much better configured for the current gloveport assignments. Thus, new gloveport combinations should be tried early on in the search. Strategic swapping of components during a new gloveport trial could also be useful to increase new gloveport assignment acceptance if the combination is indeed more favorable.
- *Components must be oriented toward the gloveports or robot(s) to be accessible* – For components that cannot be accessed from any direction, reachability depends on both position and orientation. A closer location may be less favorable than a farther location if the latter’s rotation makes access nearly impossible. Thus, when trying a new location, adjusting the component’s orientation is also beneficial to get a more accurate evaluation of the translational move on the configuration.

From these considerations, we want to allow gloveport and component swapping early in the search, even if acceptable new configurations are found from translational moves. Secondly, for each new trial configuration generated by the pattern, we allow a number of rotations at that position.

To make the search slightly less deterministic, the component pairs selected for swapping is random. Additionally, the trial rotations after a pattern translation are random since it is not easy to define the “best” rotation direction relative to all operators and

component collisions. We also incorporate random component ordering for translational moves as done by Yin [2000].

Thus, our algorithm is a hybrid approach. Knowledge or heuristics can make the search more efficient, but can only help so much. At the end of this knowledge, stochastic operations take over. This repeating cycle of deterministic and stochastic operations should lead to good final configurations.

8.3 ALGORITHM FRAMEWORK

In light of these considerations and extensions, we implement our algorithm as follows:

- Generate an initial layout L_0
- Evaluate $f(L_0)$
- Set the initial translation step size
- While the stopping criteria are not met:
 - For each component that can be translated
 - Explore a new location using a pattern and generate layout L_1
 - If rotations are applicable
 - For the number of rotation trials, generate a random orientation
 - If $f(L_1) < f(L_0)$, accept L_1 as the current configuration ($L_0 = L_1$)
 - Else revert back to the previous L_0
 - If rotations are not applicable
 - If $f(L_1) < f(L_0)$, accept L_1 as the current configuration ($L_0 = L_1$)
 - Else revert back to the previous L_0

- If swapping is allowed
 - If human is present, randomly generate gloveport swaps
 - For the number of trials, perform a swap
 - If $f(L_I) < f(L_0)$, accept L_I as the current configuration ($L_0 = L_I$)
 - Else revert back to the previous L_0
 - Randomly generate component swaps
 - For the number of trials, perform a swap
 - If $f(L_I) < f(L_0)$, accept L_I as the current configuration ($L_0 = L_I$)
 - Else revert back to the previous L_0
- If too few moves are accepted in the above loops
 - Reduce the translation step size
- End when stopping criteria met

The following sections discuss the implementation of the above loops. The EPS algorithm is coded in CLIPS. Implementation is complicated by the task plan as the sequence of operations and movement of materials must be incorporated. This requires references to the *Move* functions during various stages of the optimization loop.

8.3.1 Parameters

Various patterns can be used for translational moves. The pattern matrices reflect the permitted move directions for each component and the generally preferred search strategy [Yin, 2000]. For our examples, moves are typically constrained to the x-y plane (i.e. glovebox floor). We select a pattern that adequately searches the area about the current component location. The same pattern is used for all components and (non-

human) operators. We use a basic formulation although more advanced work exists addressing the effects of using different patterns and the order in which they are applied [Yin, 2004][Aladahalli, 2007]. There is also flexibility for three-dimensional extensions.

We end the search using a minimum step size. This level typically reflects the known computational accuracy or the required solution accuracy [Torzcon, 1997]. Our search terminates when the step size is below 1 cm – the desired resolution of our component locations. This condition could vary based on the accuracy of component geometry.

The decrease in step size varies with the current step size. Early on in the search process, the step size is large – typically a fraction of the smallest glovebox floor dimension. Toward the end of the search, the step size is much smaller. A threshold can be set for both the early and later search stages based on the percentage of new configurations that are accepted. If the acceptance percentage is below this level, a step decrease occurs. This step decrease is smaller later in the search to allow better convergence, while larger in the early stages to prevent too many searches with step sizes that are too coarse. If enough new configurations are accepted, that step size is retained for another iteration of exploratory moves. We try a number of step and/or acceptance threshold variations in our layout trials to tune our controlling parameters.

8.3.2 Preliminary Operations

A few operations are performed before beginning the search loop. The algorithm uses a design’s task plan to initialize various slot values:

- *using* and *used_by* – Each *Move* defines an operator (*component* slot) that works with the *from* and *to* components. For each *Move*, the component *titles* are added

once to the operator's *using* slot, while the operator's *title* is added once to each component's *used_by* slot.

- *circles* – The collision model (i.e. circles) for each component is constructed.

These values are crucial for reachability and collision checks.

8.3.3 Initial Configuration

This stage performs a number of important operations. First, the *origin* (i.e. gloveport location) for each human *Move* is randomly selected. Then, the initial configuration is found by addressing each *Move* in the task plan from first to last. For each *Move*, if the robotic operator, *from* component, and/or *to* component do not have locations assigned (i.e. their *location* slot is “0 0 0”) a random x-y location is found within a feasible subset of the glovebox floor. Robots are placed first, followed by the *from* component then the *to* component. Rotatable components are assigned a random orientation about the glovebox z-axis in the range of -180 to 180 degrees. If a subsequent *Move* involves a robot or component that is already placed, the location is not modified. The exception is when the component is the object (i.e. *noun*) moved. In this case, a new location is found for the component and appended to its *location* slot. As each *Move* is stepped through, the locations of processed materials are also tracked. Their locations are populated based on the movement and/or locations of the components containing and/or processing them. The final configuration representation is a list containing all the flows used in the design with populated *location* slots.

8.3.4 Evaluation

When the CLIPS evaluation function receives a configuration, it cycles through each *Move* to determine the initial location of materials. For each *Move*, the materials and their locations are stored in the *Move*'s *materials* slot. These values are retrieved when the dose calculation is performed. The function returns objective values for the configuration: the cycle time, the human dose, and the robotic dose. The constraint checking is performed separately from the evaluation and returns the constraint penalties for boundary violations, collisions, and reachability. When comparing configurations, results from each configuration's evaluation and penalty assessment are utilized. The transformed/penalized objective function is given as

$$f_P = W_{CT} N_{CT} f_{CT} + W_{HD} N_{HD} f_{HD} + W_{RD} N_{RD} f_{RD} + P_T \quad (8.1)$$

where *CT* refers to the cycle time, *HD* to the human dose, *RD* to the robotic dose, *W* are weighting factors, and *N* are normalization factors. Objective normalization factors are typically the maximum value seen during the optimization process. The penalty term P_T is given by Equation 7.17.

8.3.5 New Configuration

One iteration of the optimization algorithm consists of trying new locations and/or rotations for each component and performing any necessary swaps. Each component cycles through its exploratory moves one at a time, and the component ordering is randomized. Other possible criteria for selecting the placing/modification order include highest demand and interactions with already placed components [Tay, 1996][Barral, 2001]. New locations are tried for each location in a component's *location* slot. Any other

components or materials that have the location to modify in their *location* slot are also changed. Thus, frequently moved components or those that interact more with other components are modified more during an iteration. Thus, direct calculation and pre-computation of component demand and interaction is not needed. Additionally, random component placement removes bias and the additional computation needed to determine the “best” next component to place.

At each new location, a series of rotations is also tried if the component’s access location is not reachable from all directions. These perturbations are about the component’s *theta* value at the current best location. A random value between -180 and 180 degrees is added to the current *theta* value. Before evaluation, this modified *theta* value is forced within the same degree range for easier interpretation. The new location and/or orientation is accepted if the configuration is better.

8.3.6 Swapping

If allowed, swapping is performed after the translational and/or rotational moves. If a human operator is present, gloveport swaps are performed before component swaps. Otherwise, only component swaps occur. The number of swaps is typically a fixed number.

A gloveport location is the *origin* for a human *Move*. During a swap, a random human *Move* is selected. The *Move origin* is queried and replaced with a different gloveport location from the glovebox’s *port* slot. The swap is accepted if it is better. Otherwise, the pre-swap gloveport location is retained.

During component swaps, two random components are selected. Then a location is randomly selected from each component and the two locations are swapped. The

location swap is also performed for any other flows that contain either or both of these locations. The swap is accepted if better. Otherwise, the pre-swap locations are restored.

8.4 SUMMARY

The extended pattern search algorithm presented in this chapter has been implemented with the model presented in Chapter 7. Both the model and EPS algorithm are compatible with the state of conceptual design knowledge and are designed for quick computations. This implementation allows for the evaluation of many conceptual layouts in a short time and presents another facet of automated design. The EPS algorithm is also simple to implement, run, and adapt to new modeling information. In the next chapter, we provide layouts generated by the EPS algorithm and discuss its general abilities and benefits to the design process.

Chapter 9: System Layout Examples

The extended pattern search algorithm was applied to several glovebox layout scenarios. The influence of dose and time minimization on the layouts is clearly seen. The constraints drive the layout toward configurations that allow the operator ample room to maneuver around the glovebox. The EPS algorithm efficiently produces good configurations and provides a more efficient means of layout generation than trial-and-error in addition to providing evaluation metrics.

9.1 GENERAL EXAMPLE

The first layout trial configures the task plan listed in Table 9.1. Currently, only the *Moves* in the task plan are used to construct a layout. Figure 9.1 shows the location information generated for the initial configuration based on this task plan. The component name is followed by its *location* slot in parentheses followed by its *theta* value (which is “0” for most components). Most components also have one location (i.e. they are not moved during the process). However, for example, *Med-crucible-1* has three location triplets: the starting (first) location “90 66 0” on the glovebox floor, when it is inside *Furnace-1* at “106 86 0” (the same location as *Furnace-1*), and its new (third) location “85 46 0” when it is moved from *Furnace-1* and placed back on the glovebox floor. The processed flows, *Small-mat-2*, *Med-mat*, and *Compos-1*, have locations related to the containers or apparatuses they are located in or transferred to. For example, *Small-mat-2* has five locations: storage in *Med-cont-1*, transfer to *Med-crucible-1*, transfer to *Furnace-1* (when *Med-crucible-1* is moved there), transfer from *Furnace-1* (inside *Med-crucible-1* at its third location), and finally its movement to *Press* (as part of *Compos-1*).

During layout evaluation, the algorithm selects which flow locations to use for a given *Move* based on task plan information.

Move	Operator	From	To	Description
1	SIA10	Med-cont-1	Med-cont-1-Lid	Remove Med-cont-1 lid, place lid in space
2	SIA10	Med-cont-1	Med-crucible-1	Move Small-mat-2 between containers
3	Man	Med-cont-2	Med-crucible-1	Move Med-mat between containers
4	Man	Furnace-1	Furnace-1	Open Furnace-1
5	SIA10	Med-crucible-1	Furnace-1	Load Med-crucible-1 into Furnace-1
6	Man	Furnace-1	Furnace-1	Close Furnace-1 before running
7	Man	Furnace-1	Furnace-1	Open Furnace-1 after running
8	SIA10	Furnace-1	Med-crucible-1	Remove Med-crucible-1 from Furnace-1
9	Man	Med-crucible-1	Press	Move Compos-1 to Press

Table 9.1: The *Moves* from the general example task plan.

INITIAL CONFIGURATION:

SIA10 (47 105 0) 0
Furnace-1 (106 86 0) -91.0
Press (44 75 0) -84.0
Med-cont-1 (61 88 0) 0
Med-cont-2 (86 79 0) 0
Med-crucible-1 (90 66 0 106 86 0 85 46 0) 0
Med-cont-1-Lid (61 88 0 42 67 0) 0
Small-mat-2 (61 88 0 90 66 0 106 86 0 85 46 0 44 75 0) 0
Med-mat (86 79 0 90 66 0 106 86 0 85 46 0 44 75 0) 0
Compos-1 (106 86 0 85 46 0 44 75 0) 0

Figure 9.1: The initial configuration for the layout in Figure 9.2.

We can estimate the exposure to both the human and robot for this process by choosing arbitrary photon sources. Using the information and assumptions from Section 7.2.2, we can calculate the *exposure rate* from each monoenergetic photon source as

$$\dot{X}(r) = \frac{\mathcal{R}_x(E)S_p}{4\pi r^2} = \frac{S_p}{4\pi r^2} E \left(\frac{\mu_{en}(E)}{\rho} \right)_{air} = \frac{C}{r^2} \quad (9.1)$$

where S_p equals the source strength/photon emission rate and the appropriate conversion factors are selected to get the desired exposure unit. Using Equation 9.1, we can build reasonable arbitrary dose constants for *Small-mat-2* and *Med-mat*. In this example, the dose constant value for *Small-mat-2* is set to twice that of *Med-mat*. The needed parameters are in Table 9.2 (the absorption coefficients are from Shultis [2000]).

Exposure parameters	
<i>0.8 MeV Photons:</i>	
Source strength	1000 photons / s
Absorption coefficient	$2.882 \times 10^{-2} \text{ cm}^2 / \text{g}$
C(0.8 MeV)	$2.939 \times 10^{-10} \text{ Gy cm}^2 / \text{s}$
<i>1.5 MeV Photons:</i>	
Source strength	1000 photons / s
Absorption coefficient	$2.547 \times 10^{-2} \text{ cm}^2 / \text{g}$
C(1.5 MeV)	$4.870 \times 10^{-10} \text{ Gy cm}^2 / \text{s}$
Total C (for <i>Small-mat-2</i>) = $7.809 \times 10^{-10} \text{ Gy cm}^2 / \text{s}$	

Table 9.2: The parameters used for the exposure calculation.

In Figure 9.2, the initial component locations are in wire-frame while the final locations are the solid objects. The access locations of boxed-shaped components are shown as red spheres. A four point pattern using a constant step along the positive and negative x and y axes was used. The algorithm performed four trial rotations for box-shaped components. The glovebox has two gloveport pairs all on a single side of the glovebox. A 15 cm buffer is used for detecting components in the way of paths. The final layout evaluation and *Move* gloveport locations are shown in Table 9.3 and Table 9.4.

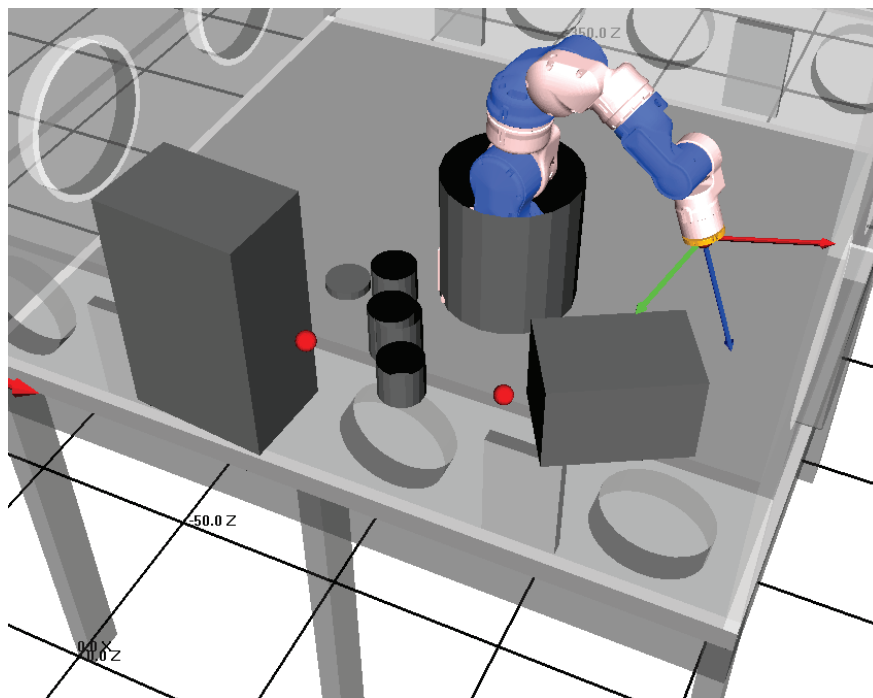
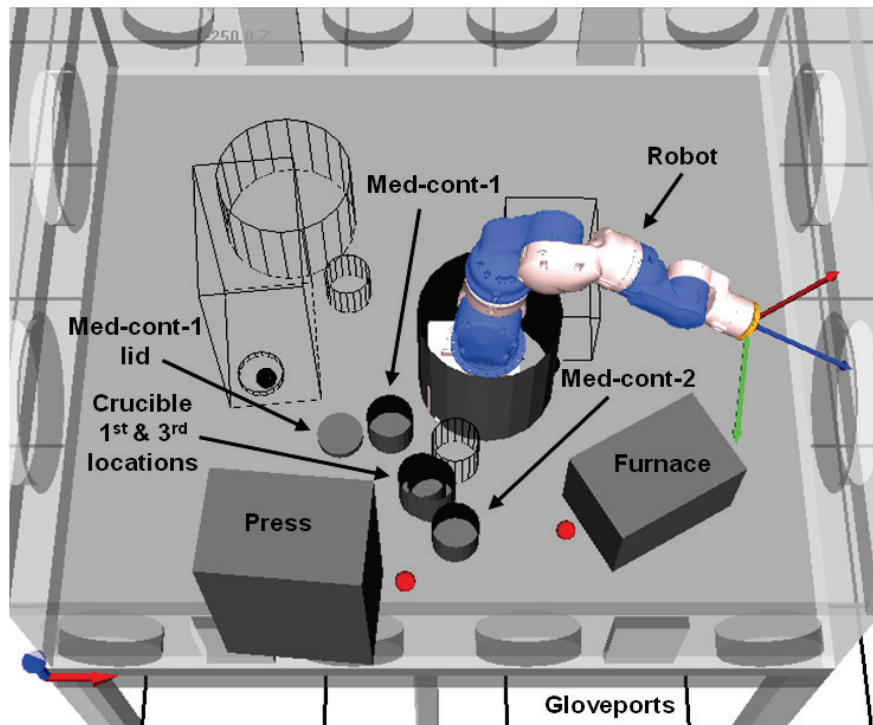


Figure 9.2: Two different views of the layouts for the task plan in Table 9.1.

Statistics:	
<i>New configurations</i>	2560 (+ 1440 swaps)
<i>Cycle time</i>	52.05 s
<i>Human exposure</i>	0.23×10^{-10} Gy
<i>Robot exposure</i>	0.29×10^{-10} Gy
<i>Collision penalty</i>	0.0
<i>Reach penalty</i>	0.044
<i>Bounds penalty</i>	0.0

Table 9.3: The results from the layout generated in Figure 9.2.

Move	Initial	Final
3	(18 0 20)	(100 0 20)
4	(146 0 20)	(100 0 20)
6	(146 0 20)	(100 0 20)
7	(100 0 20)	(100 0 20)
9	(100 0 20)	(100 0 20)

Table 9.4: The initial and final human gloveport locations.

The layout in Figure 9.2 represents the best case where human operations can be completed from a single gloveport pair. The algorithm also commonly finds a layout where most operations are performed from one gloveport pair with one *Move* executed from the other gloveport pair. Such an example is seen in Figure 9.3. Hybrid workcells with different human/robotic task allocation could also be generated then configured to determine the best workcell task allocation. This topic is addressed in the next section.

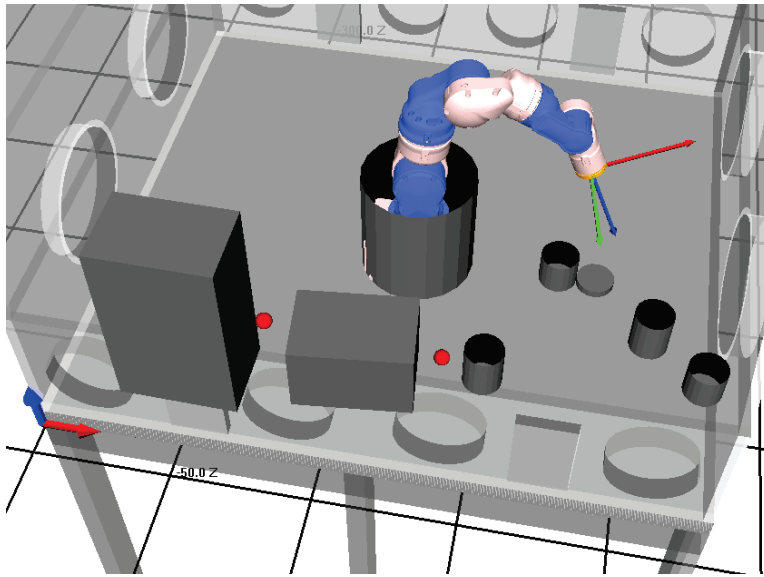


Figure 9.3: Another layout generated for the task plan in Table 9.1.

Various general properties of generated layouts are seen in these configurations. These properties and their influences are summarized below:

- Time minimization:
 - Higher component density
 - *From/to* components are next to one another, if possible
 - Human work is performed using a minimal number of gloveports
- Dose minimization:
 - Objects that contain nuclear materials at some point in the process are on the outskirts of the robotic and human workspaces
 - These components are not as far from the robot (lower weight for robot dose)
- Component grouping:
 - Objects only used by the robot are near to it

- Objects only used by the human are near to it
- Objects used by both operators are in the overlap of their workspaces
- Component rotations:
 - Objects are rotated toward one or more operators
 - Objects involved in a *Move* containing a rotatable component (i.e. a component with a “front”) are not located behind the component
- Cleared workspace influenced by:
 - Reachability check:
 - Penalty for objects near or in straight-line path between operator and component access point
 - Result: typically no objects in direct reach path
 - Minimize time between *from/to*:
 - Remove collisions with other components while bringing the *from* and *to* components closer together
 - Result: typically no objects between *from* and *to*
- Implicit objectives/constraints handled by formulation:
 - Minimum separation between components: from collision and path checks
 - No objects located between *from* and *to*
 - Cleared workspace helps for maintainability and operator self-movement
 - Non-negative values for via points on paths from operator to target: penalty for components being out-of-bounds and preference for algorithm to move components toward workspace outskirts

These are all the types of checks that a human would consider when configuring a workcell. The computer incorporates them automatically.

In general, the implemented objectives and constraints produce layouts that are favorable for manipulation. Additionally, although the locations are explicitly two-dimensional, there are various three-dimensional assessments. These include three-dimensional path generation, access and operator locations, and distance calculations. For example, in Figure 9.3, the algorithm finds that the press access point is above the furnace height and thus the press reach is not blocked by the furnace location. Utilization of space above components is beneficial, especially for reaching and transfer movements. In general, the layouts are feasible without requiring many motions that reach over components. Thus, if the workspace is clear above the components, the higher the likelihood that the layout is actually feasible when simulations are performed.

9.2 TASK ALLOCATION

The next example uses the same task plan in Section 9.1 but generates two layouts, one for a fully manual (i.e. human) workcell (Figure 9.4) and one for a fully robotic workcell (Figure 9.5). Similar algorithm parameters were used as in Section 9.1.

From Table 9.5, the robot takes longer to execute the process. This is most likely due to the robot velocity which is set to half that of the human. The time is not twice the human time because the robot layout packing density is typically higher and the workspace is more flexible than the human constrained to gloveports. In both cases, the component layout resembles the shape of the operator's workspace – half circle-shaped for the human and full circle-shaped for the robot. Access locations are oriented in the direction of the operator's center. We expect to see these types of layouts based on intuition.

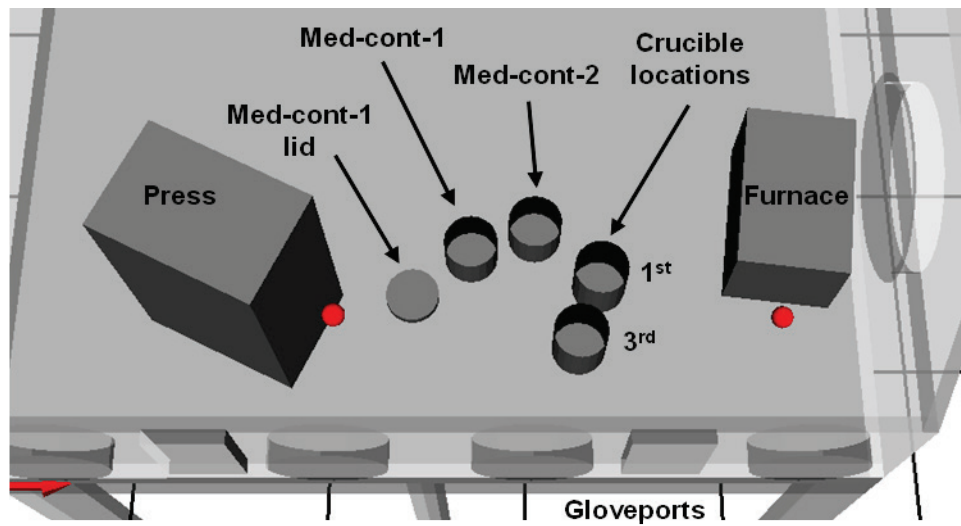


Figure 9.4: The initial and final locations for a manual workcell.

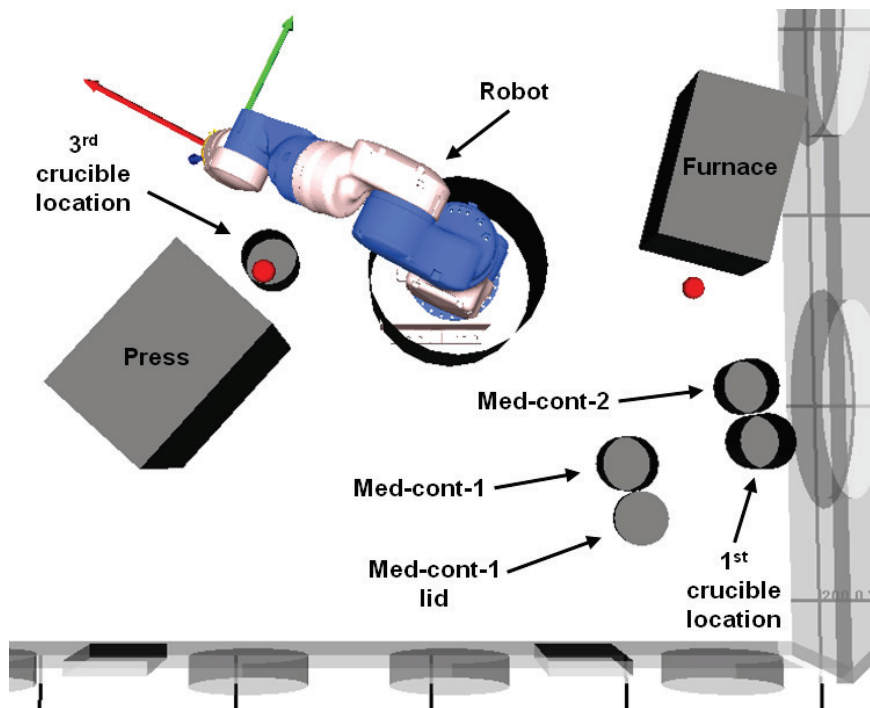


Figure 9.5: The initial and final locations for a robotic workcell.

	Manual	Automated
<i>Configurations</i>	2820 (+ 2040 swaps)	2496 (+ 750 swaps)
<i>Cycle time</i>	70.77 s	84.33 s
<i>Exposure</i>	0.26×10^{-10} Gy	0.26×10^{-10} Gy
<i>Collision penalty</i>	0.0	0.0
<i>Reach penalty</i>	0.018	0.0
<i>Bounds penalty</i>	0.0	0.0

Table 9.5: Comparison of objective and penalty values for different workcells.

More runs of each workcell are needed to better estimate the preferred concept. In general, optimization can help evaluate the changes in performance that occur from modifications in workcell task allocation. For example, starting from the manual baseline in Figure 9.4, some tasks could be reassigned to robotics, then the workcell reconfigured. Performance comparisons between the workcells would help estimate the usefulness or benefit of the design change.

9.3 AMERICIUM CONVERSION

This example contains similar tasks needed for Am conversion operations using Am oxalate received from the LANL CLEAR line (see Section 5.2.3). For this layout, a more precise human dose calculation is made – the dose constants C are not arbitrarily selected. Instead, the information in Table 9.6 is used in Equation 7.12 to calculate C for 50 g of Am-241. A select number of photons were used for the calculation with data taken from the National Nuclear Data Center [NNDC, 2013]. The response functions are for effective dose with an anteroposterior (AP) beam from Shultis [2000].

Am-241 (m = 50 grams)	
<i>Half-life:</i>	432.6 years = 1.364×10^{10} s
<i>Atomic weight:</i>	243.06 g/mol
<i>Selected photons:</i>	13.9 keV (x-ray), f = 37%
	59.5 keV (gamma), f = 35.9%
<i>Response functions:</i>	$R(E=14 \text{ keV}) = \sim 0.015 \text{ MeV} = 0.129 (10^{-12} \text{ Sv cm}^2)$
	$R(E=60 \text{ keV}) = 0.060 \text{ MeV} = 0.382 (10^{-12} \text{ Sv cm}^2)$
<i>Total dose constant C:</i>	$C(13.9 \text{ keV}) = 0.0344 \text{ Sv cm}^2 \text{ s}^{-1}$
	$C(59.5 \text{ keV}) = 0.0991 \text{ Sv cm}^2 \text{ s}^{-1}$
	Total C = 13350 mrem cm² s⁻¹

Table 9.6: The values used to calculate the dose constant C for Am-241.

The layout results are seen in Figure 9.6, Table 9.7, and Table 9.8. The run time is for an HP xw9300 Workstation running Windows XP with an AMD Opteron 252 processor rated at 2.59 GHz.

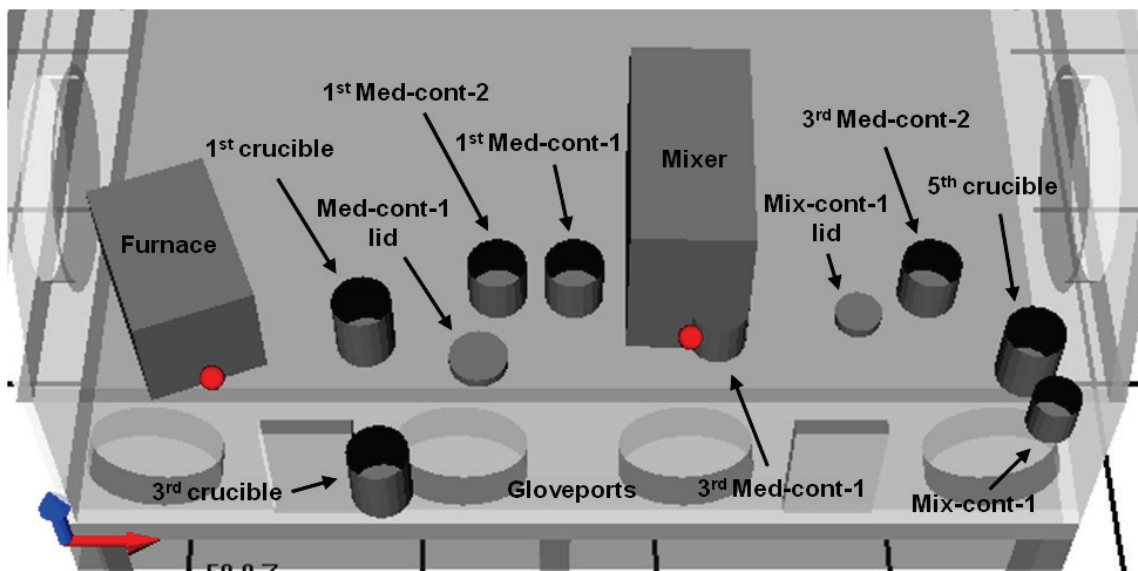


Figure 9.6: The final layout using manual processing for Am conversion.

Move	Initial Origin	Final Origin	From	To
1	(100 0 20)	(64 0 20)	Med-cont-2	Med-crucible-1
2	(18 0 20)	(100 0 20)	Med-crucible-1	Med-cont-2
3	(100 0 20)	(18 0 20)	Furnace-2	Furnace-2
4	(18 0 20)	(18 0 20)	Med-crucible-1	Furnace-2
5	(100 0 20)	(18 0 20)	Furnace-2	Furnace-2
6	(146 0 20)	(146 0 20)	Mix-cont-1	Mix-cont-1-Lid
7	(64 0 20)	(18 0 20)	Furnace-2	Furnace-2
8	(64 0 20)	(18 0 20)	Furnace-2	Med-crucible-1
9	(64 0 20)	(100 0 20)	Med-crucible-1	Mix-cont-1
10	(146 0 20)	(146 0 20)	Mix-cont-1	Med-crucible-1
11	(18 0 20)	(100 0 20)	Med-cont-1	Med-cont-1-Lid
12	(18 0 20)	(100 0 20)	Med-cont-1	Mix-cont-1
13	(146 0 20)	(146 0 20)	Mix-cont-1	Med-cont-1
14	(100 0 20)	(146 0 20)	Mix-cont-1-Lid	Mix-cont-1
15	(64 0 20)	(100 0 20)	Mixer-1	Mixer-1
16	(64 0 20)	(100 0 20)	Mix-cont-1	Mixer-1
17	(100 0 20)	(100 0 20)	Mixer-1	Mixer-1

Table 9.7: The task plan using manual/human labor for Am conversion.

Performance:	
<i>Configurations</i>	4860 (+ 3500 swaps)
<i>Cycle time</i>	232.52 s
<i>Exposure</i>	1399.64 mrem
<i>Collision penalty</i>	0.56
<i>Reach penalty</i>	2.05
<i>Bounds penalty</i>	0.4
<i>Iterations</i>	45
<i>Run time</i>	2.42 hours

Table 9.8: The final performance of the manual Am conversion concept.

The estimated dose received by a worker manually performing all the necessary conversion tasks is around 1.4 rem. The actual operational dose would be smaller since shielding would be considered and introduced and the task plan would be optimized. Even then, a large human dose is likely and concepts utilizing robotics may be necessary

for this process. Comparing this design with layouts using different task allocation would give perspective to the effectiveness of implementing robotics for selected tasks.

This problem is the largest run by the EPS algorithm. Each iteration, 108 new configurations are generated from translational and rotational moves. The evaluation is quite complex as it is influenced by 17 *Moves*. A large number of gloveport swaps were accepted early on in the optimization, as evidenced by Table 9.7. The result is longer sequences of operations from the same gloveport pair, particularly *Moves* 9 – 17. A total of 50 gloveport swaps were tried when allowed, but this is a small fraction of the total possible gloveport origin combinations for all *Moves*. More swaps are necessary to produce lower cycle time layouts, but this becomes more computationally expensive and the goal is to achieve a good design without enumerating all possibilities.

In this example, the algorithm has a difficult time finding locations for two components in the final layout: the third location of *Med-cont-1* and the third location of *Med-crucible-1*. The trade-offs between dose and penalties coupled with the gloveport constraint for human operations could be contributing factors. The component density becomes quite large as the optimization progresses and free space exists later on but it is closer to the human and would present higher doses. Depending on the trade-offs, one or more components may get trapped in an inferior or infeasible location. However, in the smaller-sized problem in Figure 9.4 and Table 9.5, the algorithm successfully converges on a feasible solution for completely manual operations. In general, the problem scale here is significantly large, complicating evaluation and convergence. Developing algorithms that can handle increasing complex and large problems is a known difficulty expressed in some of the previous configuration optimization work discussed in Section 6.1.

9.4 PU ELECTROREFINING

We generate an automated workcell with a task plan similar to that in Figure 5.7. The task allocation is slightly different with two SIA5 manipulators as the material handlers. One robot handles container loading operations and the press while the other robot handles furnace operations. Arbitrary dose constants C were selected for the materials. The resulting layout is seen in Figure 9.7.

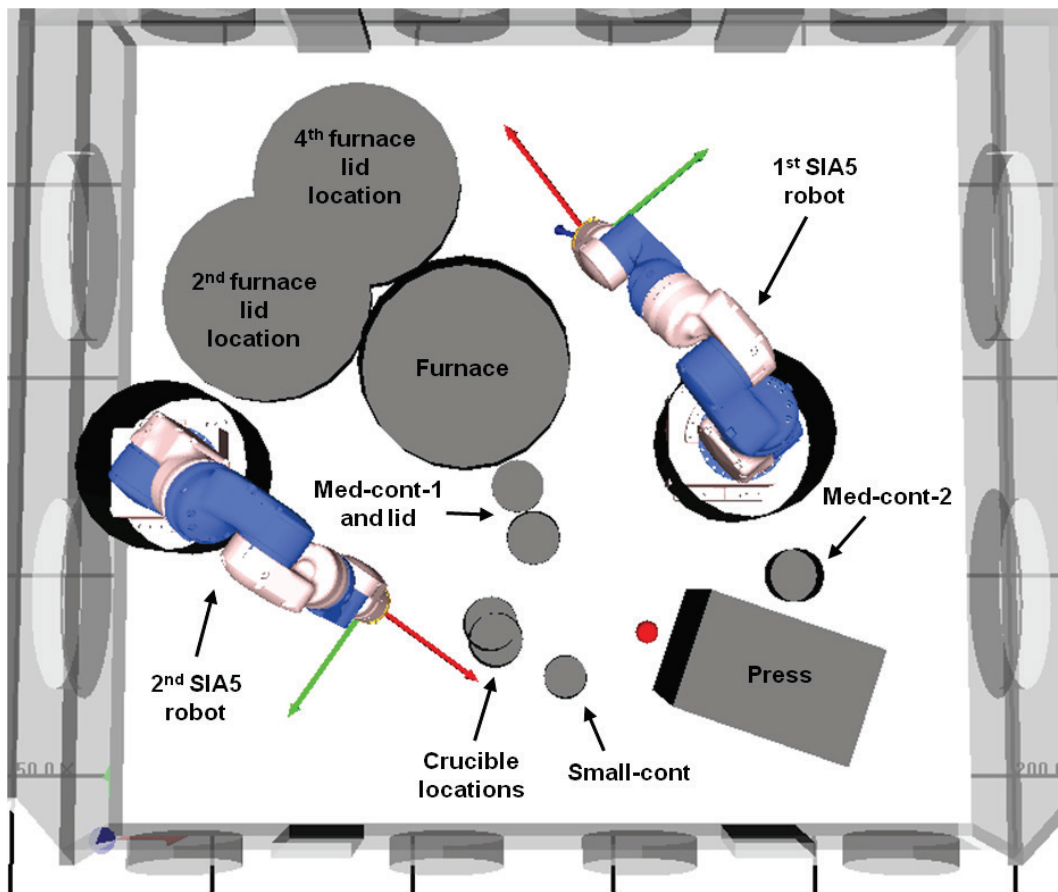


Figure 9.7: The layout for an automated Pu ER workcell.

Statistics:	
<i>New configurations</i>	4104 (+ 930 swaps)
<i>Cycle time</i>	138.65 s
<i>Total exposure</i>	414.94 Gy
<i>Collision penalty</i>	0.0016
<i>Reach penalty</i>	0.0
<i>Bounds penalty</i>	0.0

Table 9.9: The performance of the automated ER workcell.

In Figure 9.7, the components handled by a particular robot are tightly packed about its base. The crucible, which is the only component used by both robots, is located in the overlapping workspace for each of its two shared locations. Thus, the EPS algorithm is able to efficiently configure all workspace constraints. For such a layout without a human worker, the influence of penalties is not as much of a factor during the optimization process. That is, it is easier to converge on a penalty-free configuration in this case. The minimal collision penalty in the final layout is probably in the collision buffer region. The problem is larger than that presented in Figure 9.5 and Table 9.5 and the algorithm similarly converges well to a feasible solution. Thus, this example also demonstrates that the layout algorithm is applicable to configuring robotic workcells in open space since the presence of the glovebox does not particularly constrain this design.

9.5 ALGORITHM PERFORMANCE

Due to the stochastic algorithm properties, the generated layouts vary between multiple runs. Table 9.10 presents the results from six runs of the example in Section 9.1. For each design, the cycle time (CT), the human dose (HD), and the robotic dose (RD)

are shown with their respective penalties (bounds, collisions, and reach). The units are the same as in other runs for this example.

Run	Start CT	End CT	HD	RD	Start Penalties	End Penalties
1	138.50	95.41	0.21	0.29	(2.74, 78.76, 0)	(0, 0.27, 0)
2	110.07	88.63	0.20	0.39	(1.08, 71.32, 0)	(0.001, 0.34, 0.002)
3	168.49	116.97	0.36	0.49	(1.73, 89.62, 0)	(0.03, 0.37, 0)
4	90.71	52.05	0.23	0.28	(1.24, 46.10, 0)	(0, 0.04, 0)
5	151.88	80.38	0.31	0.28	(0.2, 50.81, 0)	(0.12, 0.25, 0)
6	180.40	86.14	0.25	0.31	(0.21, 67.39, 0)	(0, 0.20, 0)

Table 9.10: The results from six runs of the same task plan.

Run	Iterations	Run Time (min)
1	50	33.1
2	40	29.1
3	34	23.9
4	40	30.7
5	41	33.0
6	59	39.1

Table 9.11: The computational metrics for the runs in Table 9.10.

From this data we see that solution performance can depend on the initial layout. The start cycle times are shown along with the start penalties since cycle time tends to trend with the reach penalty. Thus, the six runs represent a range in the quality of initial layouts. A slightly more aggressive step schedule was applied during optimization to reflect that we would like to have a possibly more efficient search if we are generating a larger number of layouts. In general, there is no consistent trend in the final penalty relative to the starting configuration. The final solution quality appears to be dependent on the search quality early in the optimization. For example, all runs have similar numbers of gloveport swaps, but *Run 3*, in particular, has the largest problem with

convergence and likely suffered from a lack of quality gloveport swaps. More work is needed to address the more constrained case of human workcell layout. Without the gloveport constraint, workcells converge much easier to good solutions. In general, variations between runs are typical for these types of problems and algorithms and the optimization should be run numerous times to get a better estimate of actual design performance. As discussed below and commonly addressed in other work, there are various considerations for fine tuning numerical optimization algorithms.

Figure 9.8 shows the objective and penalty function value trends for *Run 4*. The tradeoffs between the objectives and penalties can be seen. The trends are more varied early in the search as more significant tradeoffs are possible with higher penalties, larger steps, and random swaps. As the number of iterations progresses, both the objective and penalty values begin to level out. The dose is initially low because objects are far away from the human, resulting in the large reach penalty. As the reach penalty converges to zero, the dose values start to climb to the minimum dose for a feasible (i.e. reachable) layout. Overall, the behavior is as expected.

The data for the Am conversion workcell in Figure 9.6 show similar trends (Figure 9.9). The convergence does not appear to be too rapid. This may confirm that the design became trapped in an inferior configuration/minimum.

The penalized objective function formulation works well for good control parameter selections. However, if the penalties are forced to decrease too quickly, the design may converge too soon on a particular configuration. Such configurations typically violate bounds after large steps are taken to improve reach. The violating component may become trapped after contributing such a large objective value decrease in that step. Random swaps or other possible extensions described in the next section may offset this effect.

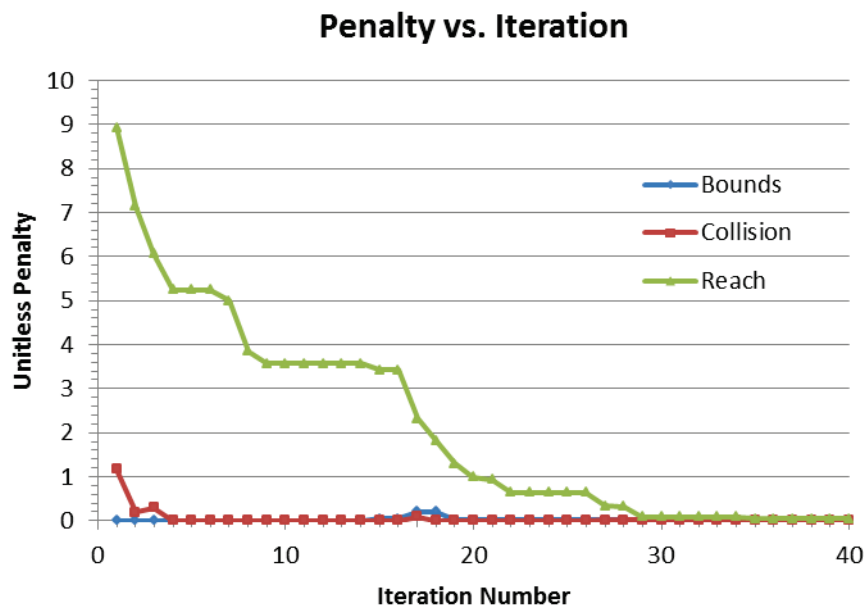
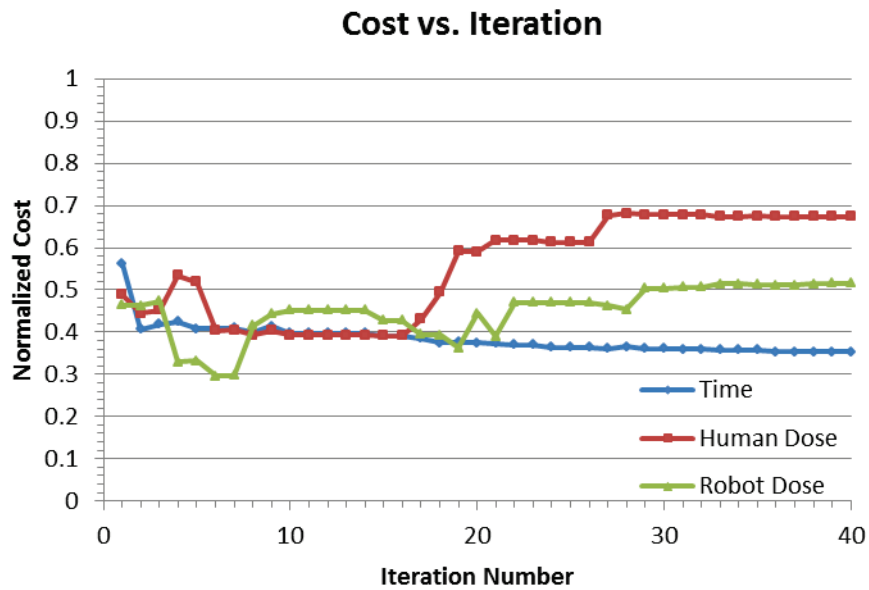


Figure 9.8: The objective and penalty function value trends for *Run 4*.

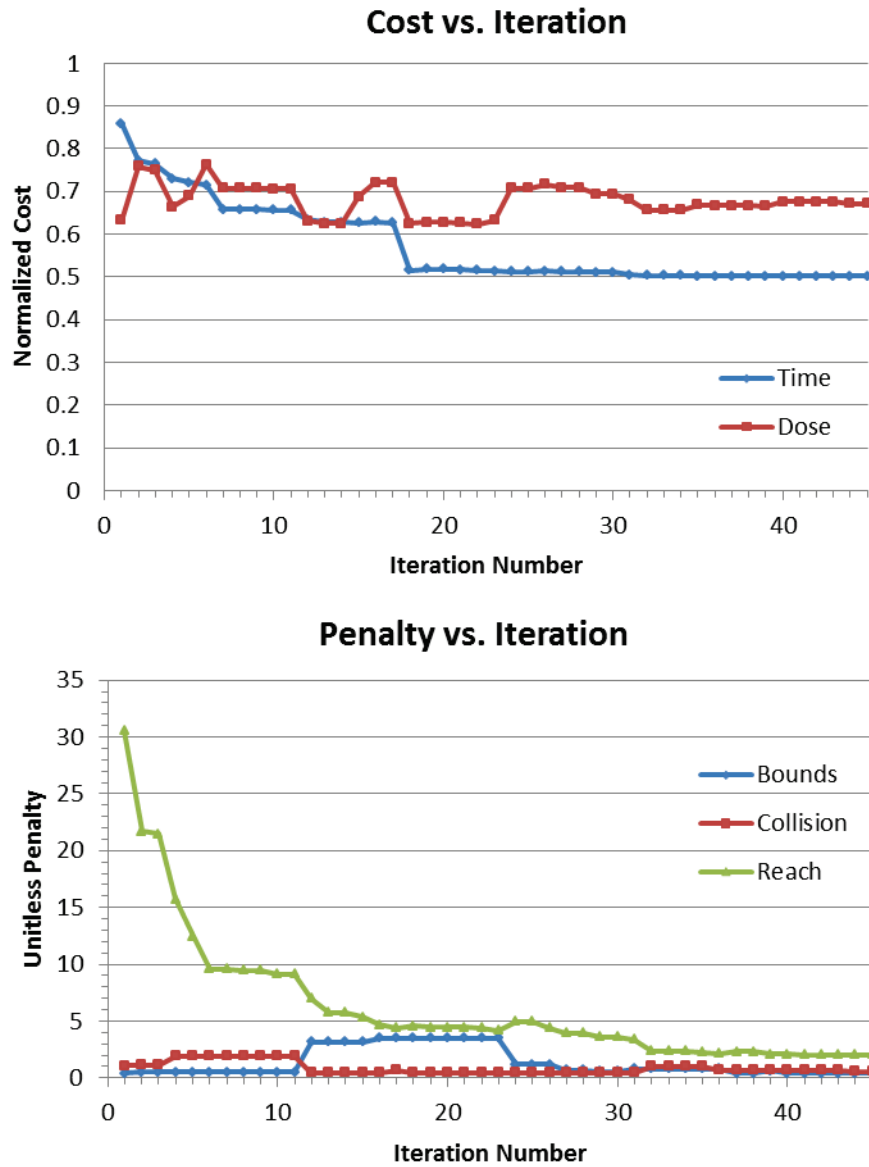


Figure 9.9: The objective and penalty function value trends for Am conversion.

This decrease effect is illustrated in Figure 9.10 for *Run 5*. The initial configuration and best results from the first two iteration cycles are shown in wireframe,

blue, and black, respectively. Due to the reach penalty (in Table 9.12), the *Press* and *Furnace* take large steps toward the human. There is now a bounds penalty because the *Press* has over-stepped two boundaries. Also, in only two iterations (with a step size of 20 cm), the components are already orienting themselves toward the human and converging toward the middle gloveports (see Table 9.13). Fortunately, the control parameters prevent pre-mature convergence and the final configuration is close to penalty-free – this is not always the case. In general, this demonstrates the importance of early moves on the final layout.

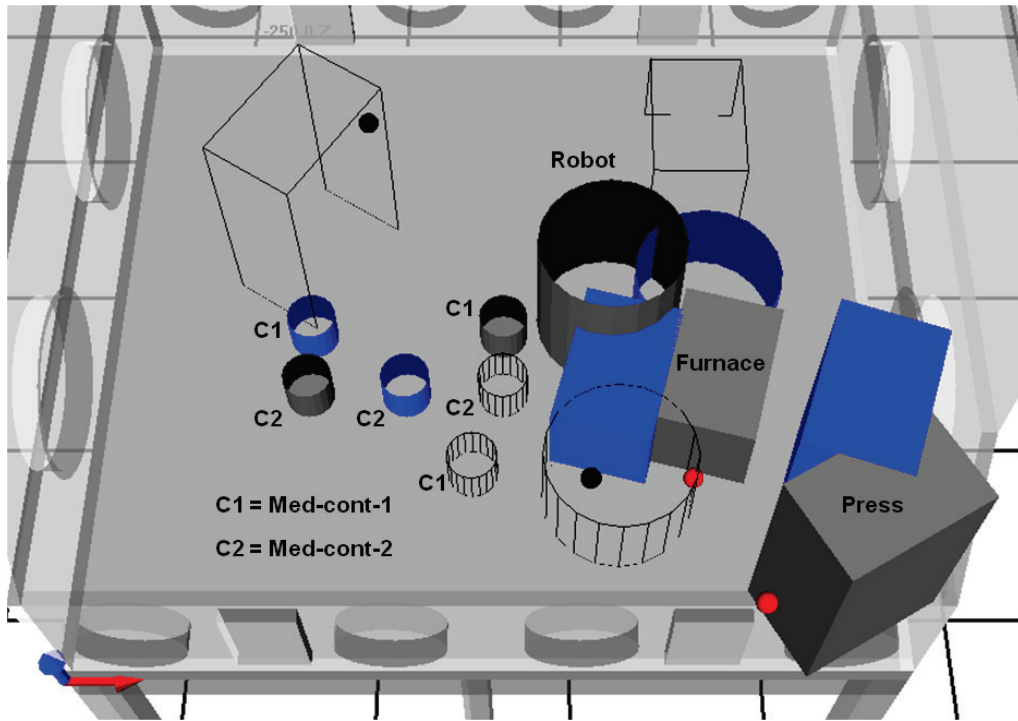


Figure 9.10: The progression of selected component locations and orientations after the first two iterations for *Run 5*.

Value	Initial	N = 1	N = 2
<i>Time</i>	151.88	128.28	120.06
<i>HD</i>	0.52	0.34	0.34
<i>RD</i>	0.83	0.36	0.52
<i>Bounds</i>	0	0.024	0.29
<i>Collision</i>	0.2	0.64	0.64
<i>Reach</i>	50.81	10.33	5.56

Table 9.12: Objective function values in the early iterations for Figure 9.10.

Move	Initial	N = 1	N = 2
3	(18 0 20)	(64 0 20)	(64 0 20)
4	(100 0 20)	(100 0 20)	(100 0 20)
6	(100 0 20)	(100 0 20)	(100 0 20)
7	(64 0 20)	(100 0 20)	(100 0 20)
9	(64 0 20)	(18 0 20)	(64 0 20)

Table 9.13: The gloveport progression for the layouts in Figure 9.10.

9.6 DISCUSSION

We focus on how the workcell's layout affects operating performance. The algorithm does not account for all objectives and constraints. For instance, the optimization does not incorporate the preference for using the left or right arm for a task. A component may be closer to a particular arm, but the arm configuration required to manipulate at that point may be unfavorable. The algorithm also does not restrict the proximity of certain component pairs. The model can increase substantially in complexity by adding such information. In general, the most important objectives and constraints should be incorporated for clearer result interpretation. However, as seen in examples,

some preferences and considerations can be implicitly handled by smart objective and constraint choices.

More work is necessary to determine the optimal parameters for a sufficient search of the solution space and convergence to an optimal solution. In most cases, a coarse step size decrease followed by a finer decrease achieved good layouts, especially when the decrease only occurred when no better solutions were found that iteration loop. This may lead to unnecessarily long stays at a particular step, but incorporating thresholds where the step decrease occurs when the percentage of accepted solutions is too small sometimes led to pre-mature convergence. Thus, setting step parameters to allow the algorithm to slowly “run its course” tended to produce good results each time.

Other penalty formulations based on Equation 6.3 were also applied with mixed results. When the step size was larger, squaring the penalties or multiplying them by a constant appeared to drive one or more components to locations that were not easily moved from (and were typically infeasible or inferior). Objective normalization was also performed using the maximum value seen in each iteration loop. A significant difference in results was not seen. The selected pattern does not appear to influence performance. However, the effect of stochastic moves on global convergence is uncertain. In general, we identified adequate parameters to produce good results, but more tests are needed to determine the exact influence of control parameters, exploratory moves, and initial configurations on the results.

Some additional extensions could be implemented to improve search effectiveness and design quality:

- Combining gloveport swaps with a simultaneous component swap
- Trying a few component rotations for each component swap
- Incorporating a “step jump” later in the process with subsequent swaps, and

- Dynamic penalty calculations.

These could particularly help alleviate the issues described in the previous section in regard to configuring human workcells. Also, the design occasionally gets trapped in a configuration where one component blocks several other components or is clearly out of reach. This could result from a lack of open space to move a component to as other components have been moved closer to the operator first and no space now exists for that possibly larger component. This may result from the random component modification order or the initial configuration. Using the above extensions, especially moves or penalties that adapt based on the current configuration properties, could address these and other issues.

There are other factors that should be considered for layouts but may be better suited once a layout is optimized. In particular, the monetary costs are important in concept selection. For example, when implementing ALARA, protection means must be economically feasible considering operating and equipment cost and the expected net benefit. Thus, the evaluation performed during layout optimization provides an indication of concept feasibility and a starting point for more involved evaluations.

9.7 SUMMARY

In our final layouts, we see expected qualities for the given objectives and constraints. Although conceptual, these layouts provide enough information to determine to a greater extent whether a generated concept is actually feasible. The EPS algorithm generates layouts faster than common stochastic techniques allowing for more design evaluations and comparisons in a given time period. Our optimization model is also applicable to designs with more detailed geometric information. In regard to an existing

design, we can approximate component geometry, derive a task sequence, and optimize the layout to determine potential design changes. Thus, our layout generation tool is useful at various abstraction levels.

Chapter 10: Conclusion

This work presented an implemented framework for automated conceptual design of manufacturing workcells in radioactive environments. The key automated design tasks include the creation of function structures and preliminary system layouts. These two tasks work hand-in-hand to address concept feasibility (particularly task compatibility of robotic and/or human labor) and provide evaluations of differently allocated workcells for a given process.

10.1 SUMMARY OF WORK

The KBS performs function-behavior-structure modeling beginning with a high-level process description. To do this, the implemented KBS modules store knowledge about the design domain. The common language not only provides the computer with a vocabulary and grammar for communicating designs, but includes the typical types of manufacturing processes, material handling tasks, processing components, and processed materials encountered in manufacturing. This functional and flow information guides which components should be included in the generic component database and the types and sequencing of operations needed to perform the overall system process with the selected components. Knowledge from engineering experience can also be incorporated including knowledge of previous solutions, heuristics, or experience working with particular components such as robotics.

All of this knowledge can be incorporated in one or more facets of the KBS, leading to generated concepts that reflect the current state of design knowledge. Given a large enough knowledge-base, this computational tool can help ensure that considerations

from multiple disciplines are accounted for and communicated in design generation. The KBS is also a powerful tool for redesign. By relaxing constraints, generated concepts can reveal possible areas of improvement through new function embodiments and task sequencing. Engineers can then revisit the current design and identify component or process modifications that make previously infeasible concepts realizable, leading to design and/or process improvements.

An initial concept may appear reasonable in theory, but a system layout provides more concrete feedback on *ultimate* feasibility. Many considerations are needed to generate a preliminary layout. Optimization is needed to give the “best” estimate of layout performance and to allow for comparison of differently allocated workcells executing the same process. Objectives and constraints must be established and the design’s task plan is vitally important as a resource for operating information/instructions and objective calculations. Thus, as with concept generation, layout optimization requires a large amount of information.

Our extended pattern search algorithm interfaced with the functionality of CLIPS provides a structure for optimizing an information-rich problem. The EPS algorithm successfully converges on a “good” layout for a given workcell task plan. These layouts illustrate how the selected labor could function in the environment. Although still conceptual in nature, the layouts provide enough information about potential locations, path clearance, and reachability to determine whether a given concept and/or layout is favorable.

This computational tool (KBS plus the EPS algorithm) embodies the objectives of this research. This work is successful in its ability to:

- Store various kinds of knowledge from multiple domains and apply this knowledge to design

- Establish a general framework for automated conceptual design that is highly flexible and extendable
- Produce feasible results for real system processes guided by qualitative and quantitative information

This tool presents a step in building an expert system for automated system design in the nuclear domain that rivals the thought processes and abilities of human design experts. Specifically, our KBS and EPS algorithm automates preliminary design (Figure 10.1).

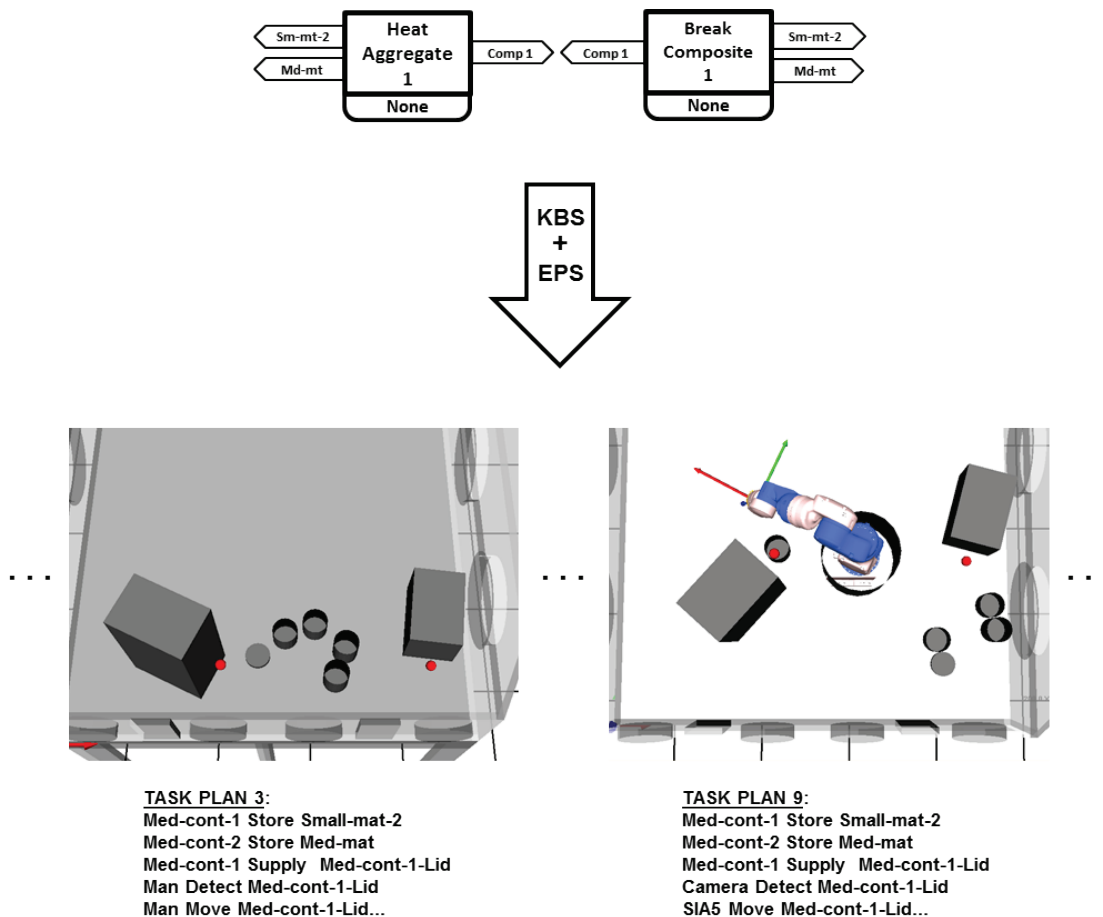


Figure 10.1: The abbreviated input and output of our conceptual design tool.

This work presents many benefits to design in the nuclear domain. In particular, the knowledge-base stores knowledge gained from the experience of many engineers. Many heuristics are passed down in nuclear work and this ensures knowledge retention that may otherwise be lost when experts leave a facility. The generation of robotic and/or hybrid workcell concepts presents solutions that may be new to the field and more favorable considering the current processing needs and administrative objectives. Concept layout and evaluation capabilities provide the means to evaluate design performance. Configured concepts may improve processing and the working environment, especially in regard to ALARA. Therefore, this tool not only contributes to research in automated conceptual design but has immediate practical applicability to existing design scenarios.

10.2 FUTURE WORK

Much future work is related to the level of incorporated knowledge and desired resolution of solutions. The general and flexible structure of the KBS is well suited for these types of changes and the EPS algorithm can handle modifications to optimization model complexity, objectives, and constraints. A number of possible extensions to this work are presented in the following sections.

10.2.1 KBS Knowledge Extensions

One always available extension is incorporating more knowledge in the KBS. This can be accomplished in a variety of ways. New specific functions could be added to the common language. The rule-base and components would need to be updated to reflect these new functions and flows. More component attributes could also be defined with

new slots for additional requirements, geometric information, compatibility information (operator-component or component-component), etc.

Some of the current qualitative requirements could also be converted to quantitative metrics. This may require more detailed parameters for some of the components. These metrics could be calculated off-line. For instance, using a full manipulator kinematic model and more complete component geometry, a simulation could produce a quantitative metric for a robot's compatibility with different representative types of door handles, locks, lids, etc. Similar measurements could be performed for multiple EEFs, and thus robotic embodiment would involve the selection of manipulator and EEF.

The knowledge-base framework is designed to be generalized, but application to a particular domain to create a domain-specific tool is possible. For instance, we could develop a list of LANL processes recognized by the KBS for specific application to LANL glovebox system designs. These specific processes would then decompose into terms from the current common language. Focusing on one domain or design case would lead to knowledge-base additions such as those described above. This type of application is better suited for adding more specific/detailed design information.

10.2.2 Concept Filtering

As described in Section 5.3, the KBS generates all possible combinations. When generating a task plan, there is no check to previous or preceding functions to make a more informed decision about the next embodiment. This means no combinations are missed but presents an exceedingly large number of concepts to search through.

The concept generation process would benefit from a technique to minimize the number of solutions presented to the designer. As currently implemented, there is no initial concept evaluation and the designer must manually choose concepts to further analyze or move on to layout optimization. Additional rules or sub-functions could be implemented to restrict or encourage certain embodiments based on previous task plan information. Campbell [2009] presents a technique for determining the designer's preference for particular rules to help search a solution tree created from a generative grammar. The full solution set could also run through an evaluation to present only a favorable subset to the designer. This evaluation could be based on the number of times tasks are switched between operators or eliminating concepts that only differ from existing concepts by the selection of their storage containers.

10.2.3 Generative Layout Optimization

The gap between concept generation and layout optimization could also be bridged by incorporating generative techniques into the optimization process. Such combined knowledge-based and optimization techniques have been tried. Heuristic rule-based techniques exist for highly deterministic layout optimization [Cagan, 2000]. Cagan [1993] implemented a shape grammar into a simulated annealing algorithm to pack geometric shapes into a container and Brown [1997] utilized generative simulated annealing for optimizing process planning.

The general approach would be to first generate an initial layout with a particular task allocation and then allow the rule-base to make changes to the task plan as layout optimization progresses. The task plan changes would include selection of new operators and components and the addition of sub-functions as needed by embodiments. A task

reassignment would be feasible if the operator (or component) meets the current task criteria. In this formulation, requirements such as *reach*, *motion*, and *dexterity* could be directly computed from the current configuration (i.e. component locations and geometries). Thus, operators could be randomly selected to see if they are compatible with the existing environment or a change could be made to the existing layout to allow a different embodiment.

Such a technique would be quite powerful. The designer would not need to search through concepts or separately configure and then compare them. In theory, this computational tool could perform all the tasks necessary to select a preliminary concept layout from a high-level functional description. CLIPS would be a compatible environment for this implementation and EPS could also be utilized for searches.

10.2.4 Layout Model Extensions

The optimization model is developed for quick objective and constraint calculations. More precise performance estimates are possible by adding more to the model. The EPS algorithm runs fast enough that, overall, more sophisticated calculations may still make the algorithm quicker than others such as SA and GAs.

At the conceptual level, there is much uncertainty regarding the nature of manipulation. Our layouts utilized fixed times for opening/closing component sub-targets. Other time estimations could be based on the operator-component pair and stored in a component's slot. The included times for optimization are only for material handling tasks. Principles from design for assembly [Boothroyd, 2001] could be utilized to calculate manipulation times for many common tasks for storage in a database that interfaces with the knowledge-base. Additionally, typical task times specific to the

process could be stored based on the process engineer's experience. Incorporating "fuzzy" principles in the formulation could also be useful here and in other estimates.

The dose derivation from Section 7.2.2.1 could incorporate more factors. In particular, shielding and buildup could be considered. To implement these, material interaction coefficients and buildup factors could be added to component instances or quickly retrieved from an associated database. This would also allow estimates for shielding from components or actual shields.

Other objectives such as ergonomics and safety could also be incorporated. Formulations for these objectives are not as straight-forward. In regard to ergonomics, there are more favorable work volumes within a human's workspace. The workspace volume could be discretized with each point receiving a score for its location relative to the human's center. Then each *Move* would have a score based on the access point locations in the calculated ergonomic workspace. A safety metric could incorporate many factors. This could include operator or reach path proximity to heat or chemical sources and the quantity of sharps handled or worked around. For clearer optimization results, it would be beneficial to select only the most important criteria for a given process.

The extension can also be made to three-dimensional layouts. Although, the z direction is considered in various calculations, it is not fully implemented. In general, the two-dimensional formulation is well suited for the considered glovebox systems. More interesting configurations are possible by including all three dimensions such as wall or ceiling-mounted manipulators or gantry systems. These configurations may also capture an operator's preference to manipulate objects at a certain height if its kinematic model is available.

10.2.5 Simulation

During layout optimization, a robot simulation sub-routine could interface with constraint checking. This simulation could check robot reach and path collisions and/or generate a rough trajectory. To speed up calculations, coarse resolution could be utilized early on to generate trajectory points for collision checking of arm configurations, with finer resolution used toward the end of optimization, if desired. Path generation and collision checking could also suggest better retraction locations after completing a move. Currently, these locations are not checked for collisions. More accurate estimates of robot movement times could also be generated through simulation.

10.3 SUMMARY

Although we have applied our KBS specifically to LANL glovebox system designs, the emphasis of this work has been on the supporting framework. This structure is designed to handle all desired types of information and design tasks during conceptual design. Although the current KBS implementation is not an all-encompassing tool for mechanical design, it can be extended or as inclusive as desired. This research brings together expertise from various fields to create a powerful computational design tool and presents techniques for advancing conceptual design automation.

References

- Abdel-Malek, K., Yu, W., and Yang, J., "Placement of robot manipulators to maximize dexterity," *International Journal of Robotics and Automation*, Vol. 19, No. 1, pp. 6-14 (2004).
- Agarwal, M. and Cagan, J., "On the use of shape grammars as expert systems for geometry-based engineering design," *AI for Eng. Design, Analysis and Manufact.*, Vol. 14, pp. 431-439 (2000).
- Aladahalli, C., Cagan, J., and Shimada, K., "Objective function effect based pattern search – an implementation for 3D component layout," *Journal of Mechanical Design*, Vol. 129, pp. 255-265 (2007).
- Altshuller, G., *Creativity as an Exact Science*, Gordon and Breach, New York, 1988.
- Arora, J.S., *Introduction to Optimum Design*, Elsevier Science, Oxford, UK, 2012.
- Arora, J.S., Elwakeil, O.A., Chahande, A.I., and Hsieh, C.C., "Global optimization methods for engineering applications: a review," *Structural Optimization*, Vol. 9, pp. 137-159 (1995).
- Baldwin, R. and Chung, M., "A formal approach to managing design processes," *IEEE Comp.*, pp. 54-63, Feb. 1995.
- Barral, D., Perrin, J.P., Dombre, E., and Liégeois, A., "Simulated annealing combined with a constructive algorithm for optimising assembly workcell layout," *International Journal of Advanced Manufacturing Technology*, Vol. 17, pp. 593-602 (2001).
- Bhangale, P.P., Agrawal, V.P., and Saha, S.K., "Attribute based specification, comparison and selection of a robot," *Mechanism and Machine Theory*, Vol. 39, pp. 1345-1366 (2004).
- Boothroyd, G., Dewhurst, P., and Knight, W.A., *Production Design for Manufacture & Assembly*, CRC Press, 2nd Ed, 2001.
- Brown, D.R. and Hwang, K.Y., "Solving fixed configuration problems with genetic search," *Research in Engineering Design*, Vol. 5, pp. 80-87 (1993).
- Brown, K.N. and Cagan, J., "Optimized process planning by generative simulated annealing," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Vol. 11, pp. 219-235 (1997).
- Bryant, C., McAdams, D., Stone, R., Kurtoglu, T., and Campbell, M., "A computational technique for concept generation," *Proc. 2005 ASME IDETC/CIE*, ASME, Long Beach, CA, 2005.

- Cagan, J., Campbell, M., Finger, S., and Tomiyama, T., "A framework for computational design synthesis: model and applications," *Journal of Computing and Information Science in Engineering*, Vol. 5, No. 3, pp. 171-181 (2005).
- Cagan, J., Shimada, K., and Yin, S., "A survey of computational approaches to three-dimensional layout problems," *Computer-Aided Design*, Vol. 34, pp. 597-611 (2002).
- Cagan, J. and Mitchell, W.J., "Optimally directed shape generation by shape annealing," *Environment and Planning B: Planning and Design*, Vol. 20, pp. 5-12 (1993).
- Campbell, M.I., Rai, R., and Kurtoglu, T., "A stochastic graph grammar algorithm for interactive search," *Proc. 2009 ASME IDETC/CIE Conference*, San Diego, CA, August 2009.
- Campbell, M.I. and Rai, R., "A generalization of computational synthesis methods in engineering design," *AAAI Spring Symposium Series*, AAAI Press, Palo Alto, CA, March 2003a.
- Campbell, M., Cagan, J., and Kotovsky, K., "The A-Design approach to managing automated design synthesis," *Research in Eng. Design*, Vol. 14, pp. 12-24 (2003b).
- Campbell, M.I., Amon, C.H., and Cagan, J., "Optimal three-dimensional placement of heat generating electronic components," *Journal of Electronic Packing*, Vol. 119, pp. 106-113 (1997).
- Carlson-Skalak, S., White, M.D., and Teng, Y., "Using an evolutionary algorithm for catalog design," *Research in Engineering Design*, Vol. 10, pp. 63-83 (1998).
- Chakrabarti, A. and Bligh, T., "A scheme for functional reasoning in conceptual design," *Design Studies*, Vol. 22, pp. 493-517 (2001).
- Chandrasekaran, B., "Representing function: relating functional representation and functional modeling research streams," *AI for Eng. Design, Analysis and Manufact.*, Vol. 19, pp. 65-74 (2005).
- Chen, Y., Liu, Z.L., and Xie, Y.B., "A knowledge-based framework for creative conceptual design of multi-disciplinary systems," *Computer-Aided Design*, Vol. 44, pp. 146-153 (2012).
- Chiu, S., "Task compatibility of manipulator postures," *International Journal of Robotics Research*, Vol. 7, No. 5, pp. 13-21 (1988).
- Cool, D.A., "US NRC discussion of options to revise radiation protection recommendations," *Annals of the ICRP*, Vol. 41, No. 3/4, pp. 313-317 (2012).
- Cournoyer, M., Trujillo, S., and Blask, C., "Microscopic analysis of glovebox glove failures and breaches," *J. of Chem. Health and Safety*, Vol. 19, No. 5, pp. 39-46 (2012).

- Dieter, G.E., *Engineering Design: A Materials and Processing Approach*, McGraw-Hill, Boston, 2000.
- Dowsland, K.A. and Dowsland, W.B., "Packing problems," *European Journal of Operational Research*, Vol. 56, pp. 2-14 (1992).
- Drira, A., Pierreval, H., and Hajri-Gabouj, S., "Facility layout problems: a survey," *Annual Reviews in Control*, Vol. 31, pp. 255-267 (2007).
- Dubey, R. and Luh, J.Y.S., "Redundant robot control using task based performance measures," *Journal of Robotic Systems*, Vol. 5, No. 5, pp. 409-432 (1988).
- Erden, M., Komoto, H., Van Beek, T., D'Amelio, V., Echavarria, E., and Tomiyama, T., "A review of function modeling: approaches and applications," *AI for Eng. Design, Analysis and Manufact.*, Vol. 22, pp. 147-169 (2008).
- Feng, S.C. and Song, E.Y., "Information modeling of conceptual design integrated with process planning," *Proc. 2000 International Mechanical Engineering Congress and Exposition*, Orlando, FL, November 2000.
- Finger, S. and Dixon, J.R., "A review of research in mechanical engineering design. Part I: descriptive, prescriptive, and computer-based models of design processes," *Research in Engineering Design*, Vol. 1, pp. 51-67 (1989).
- Fitts, P.M., "Human engineering for an effective air navigation and traffic control system," National Research Council, Washington, DC, March 1951.
- Forbus, K.D., "Qualitative process theory," *Artificial Intelligence*, Vol. 24, pp. 85-168 (1984).
- Gao, F. and Gruver, W.A., "Performance evaluation criteria for analysis and design of robotic mechanisms," *8th International Conference on Advanced Robotics (ICAR)*, Monterey, CA, pp. 879-884, July 1997.
- Genaidy, A.M., Duggal, J.S., and Mital, A., "A comparison of robot and human performances for simple assembly tasks," *International Journal of Industrial Ergonomics*, Vol. 5, pp. 73-81 (1990).
- Gero, J.S., "Design prototypes: a knowledge representation schema for design," *AI Magazine*, Vol. 11, No. 4, pp. 26-36 (1990).
- Gero, J.S. and Kannengiesser, U., "The situated function-behavior-structure framework," *Design Studies*, Vol. 25, pp. 373-391 (2004).
- Ghosh, B.K. and Helander, M.G., "A systems approach to task allocation of human-robot interaction in manufacturing," *Journal of Manufacturing Systems*, Vol. 5, No. 1, pp. 41-49 (1986).
- Giarratano, J. and Riley, G., *Expert Systems: Principles and Programming*, Thomson Course Technology, Boston, MA, 2005.

- Grignon, P.M., "Configuration design optimization method," *Dissertation*, The Department of Mechanical Engineering, Clemson University, 1998.
- Gottschlich, S., Ramos, C., and Lyons, D., "Assembly and task planning: a taxonomy," *IEEE Robotics & Automation Magazine*, Vol. 1, No. 3, pp. 4-12 (1994).
- Grecu, D. and Brown, D., "Expectation formation in multi-agent design systems," *Proc. AI in Design '00*, J. Gero, ed., Kluwer, Dordrecht, pp. 651-671 (2000).
- Hamann, T. and Vernadat, F. "The intra cell layout problem in automated manufacturing systems," *8th International Conference on CAD/CAM, Robotics and Factories of the Future*, Metz, France, August 1992.
- Harmer, Q.J., Weaver, P.M., and Wallace, K.M., "Design-led component selection," *Computer-Aided Design*, Vol. 30, No. 5, pp. 391-405 (1998).
- Hauser, J.R. and Clausing, D., "The house of quality," *Harvard Business Review*, pp. 63-73, May-June 1988.
- Helms, B., Shea, K., and Hoisl, F., "A framework for computational design synthesis based on graph-grammars and function-behavior-structure," *Proc. 2009 ASME IDETC/CIE Conference*, San Diego, CA, August 2009.
- Heragu, S.S., "Knowledge based approach to machine cell layout," *Computers & Industrial Engineering*, Vol. 17, Nos. 1-4, pp. 37-42 (1989).
- Heywood, A.C. and Armantrout, G.A., "Robotic requirements for plutonium handling automation," *IEEE Transactions on Nuclear Science*, Vol. 37, No. 3, pp. 1452-1455 (1990).
- Hirtz, J., Stone, R., McAdams, D., Szykman, S., and Wood, K., "Evolving a functional basis for engineering design," *Proc. 2001 ASME DETC*, ASME, Pittsburgh, PA, September 2001.
- Hubka, V. and Eder, W.E., "A scientific approach to engineering design," *Design Studies*, Vol. 8, No. 3, pp. 123-137 (1987).
- Hubka, V., *Principles of Engineering Design*, Butterworth-Heinemann, 1982.
- Hundal, M.S., "A systematic method for developing function structures, solutions and concept variants," *Mech. Mach. Theory*, Vol. 25, No. 3, pp. 243-256 (1990).
- Hwang, Y.K. and Ahuja, N., "Gross Motion Planning: A Survey," *ACM Computing Surveys*, Vol. 24, No. 3, pp. 219-291 (1992).
- ICRP, "The 2007 recommendations of the International Commission on Radiological Protection," *Annals of the ICRP*, Publication 103, Vol. 37, No. 2-4 (2007).
- ICRP, "Data for use in protection against external radiation," *Annals of the ICRP*, Publication 51, Vol. 17, No. 2/3 (1987).

- ICRU, "Phantoms and computational models in therapy, diagnosis and protection," *Report 48*, Bethesda, MD (1992).
- IDEF, "Welcome to IDEF.com," Knowledge Based Systems, Inc., <http://www.idef.com/Home.htm>, accessed June 2013.
- Jordan, N., "Allocation of functions between man and machines in automated systems," *Journal of Applied Psychology*, Vol. 47, pp. 161-165 (1963).
- Jung, E.S., and Shin, Y., "Two-handed human reach prediction models for ergonomic evaluation," *Human Factors and Ergonomics in Manufacturing & Service Industries*, Vol. 20, No. 3, pp. 192-201 (2010).
- Kantowitz, B.H. and Sorkin, R.D., "Allocation of functions," in *Handbook of Human Factors*, Gavriel Salvendy (Ed), John Wiley & Sons, New York, pp. 355-369 (1987).
- Kiritsis, D., "A review of knowledge-based expert systems for process planning: methods and problems," *Int. J. Adv. Manufact. Tech.*, Vol. 10, pp. 240-262 (1995).
- Kirkpatrick, S., Gelatt, Jr., C.D., and Vecchi, M.P., "Optimization by simulated annealing," *Science*, Vol. 220, No. 4598, pp. 671-680 (1983).
- Kitamura, Y. and Mizoguchi, R., "Ontology-based systematization of functional knowledge," *J. of Eng. Design*, Vol. 15, No. 4, pp. 327-351 (2004).
- Klein, C.A., "Use of Redundancy in the Design of Robotic Systems," *Robotics Research: The Second International Symposium*, Eds. H. Hanafusa and H. Inoue, Cambridge: MIT Press, pp. 207-214, August 1985.
- Kossiakoff, A., Sweet, W.N., Seymour, S.J., and Biemer, S.M., *Systems Engineering: Principles and Practice*, Wiley, Hoboken, NJ, 2011.
- Kusiak, A. and Heragu, S.S., "KBSES: A knowledge-based system for equipment selection," *International Journal of Advanced Manufacturing Technology*, Vol. 3, No. 3, pp. 97-109 (1988).
- Kusiak, A., "Process planning: a knowledge-based and optimization perspective," *IEEE Trans. on Robotics and Automation*, Vol. 7, No. 3, pp. 257-266 (1991).
- Kurtoglu, T., Campbell, M.I., Bryant, C.R., Stone, R.B., and McAdams, D.A., "Deriving a component basis for computational functional analysis," *International Conference on Engineering Design (ICED '05)*, Melbourne, August 2005a.
- Kurtoglu, T., Campbell, M.I., Gonzales, J., Bryant, C.R., and Stone, R.B., "Capturing empirically derived design knowledge for creating conceptual design configurations," *2005 ASME IDETC/CIE Conference*, Long Beach, CA, September 24-28, 2005b.

- Kurtoglu, T., Swantner, A., and Campbell, M., "Automating the conceptual design process: from black-box to component selection," *Proc. Design Computing and Cognition '08*, J. Gero and A. Goel, eds., pp. 553-572 (2008).
- Kurtoglu, T., Campbell, M., Arnold, C., Stone, R., and Meadams, D., "A component taxonomy as a framework for computational design synthesis," *J. of Computing and Inf. Science in Eng.*, Vol. 9, No. 1, 011007 (2009).
- Lander, S.E., "Issues in multiagent design systems," *IEEE Expert*, Vol. 12, pp.18-26 (1997).
- LANL, "Innovations in actinide manufacturing," *Actinide Research Quarterly*, No. 1, LA-UR-12-25866, October 2012.
- LANL, "Accelerator radioisotopes save lives: part II," *Actinide Research Quarterly*, 4th Quarter 2009/1st Quarter 2010, LALP-10-011, 2010.
- LANL, "ARIES turns 10," *Actinide Research Quarterly*, 1st/2nd Quarters, LALP-08-004, 2008a.
- LANL, "Plutonium processing at Los Alamos," *Actinide Research Quarterly*, 3rd Quarter, LALP-08-004, 2008b.
- Lauridsen, K., Christensen, P., and Kongsø, H.E., "Assessment of the reliability of robotic systems for use in radiation environments," *Reliability Engineering and System Safety*, Vol. 53, pp. 265-276 (1996).
- LeGoullon, A., "Configuration management of robotic workcells," *Master's Thesis*, The Department of Mechanical Engineering, The University of Texas at Austin, December 1997.
- Lemaignan, S., Siadat, A., Dantan, J.Y., and Semenenko, A., "MASON: a proposal for an ontology of manufacturing domain," *Proc. of IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS 2006)*, pp. 195-200, June 2006.
- Lenarčič, J. and Umek, A., "Simple model of human arm reachable workspace," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 24, No. 8, pp. 1239-1246 (1994).
- Lewis, R.M., Torczon, V., and Trosset, M.W., "Why pattern search works," *ICASE Report No. 98-57*, NASA Langley Research Center, Hampton, VA (1998).
- Lozano-Perez, T., Jones, J.L., Mazer, E., and O'Donnell, P.A., "Task-level planning of pick-and-place robot motions," *Computer*, Vol. 22, No. 3, pp. 21-29, March 1989.
- Lueth, T.C., "Automated planning of robot workcell layouts," *Proc. of IEEE International Conference on Robotics and Automation*, Nice, France, May 1992.

- Marler, R.T. and Arora, J.S., "Survey of multi-objective optimization methods for engineering," *Structural and Multidisciplinary Optimization*, Vol. 26, pp. 369-395 (2004).
- Marri, H., Gunasekaran, A., and Grieve, R., "Computer-aided process planning: a state of art," *International J. of Adv. Manufact. Tech.*, Vol. 14, pp. 261-268 (1998).
- Mata, V. and Tubaileh, A., "The machine layout problem in robot cells," *International Journal of Production Research*, Vol. 36, No. 5, pp. 1273-1292 (1998).
- McAdams, D.A. and Wood, K.L., "A quantitative similarity metric for design-by-analogy," *Journal of Mechanical Design*, ASME, Vol. 124, pp. 173-182 (2002).
- Mital, A., Motorwala, A., Kulkarni, M., Sinclair, M., and Siemieniuch, C., "Allocation of functions to humans and machines in a manufacturing environment: Part II – the scientific basis (knowledge base) for the guide," *International Journal of Industrial Ergonomics*, Vol. 14, pp. 33-49 (1994).
- Mosemann, H. and Wahl, F.M., "Automatic decomposition of planned assembly sequences into skill primitives," *IEEE Transactions on Robotics and Automation*, Vol. 17, No. 5, pp. 709-718 (2001).
- Mukherjee, A. and Liu, C., "Conceptual design, manufacturability evaluation and preliminary process planning using function-form relationships in stamped metal parts," *Robotics & Computer-Integrated Manufact.*, Vol. 13. No. 3, pp. 253-270 (1997).
- Nagel, R.L., Perry, K.L., Stone, R.B., and McAdams, D.A., "FunctionCAD: a functional modeling application based on the Function Design Framework," *Proc. 2009 ASME IDETC/CIE*, ASME, San Diego, CA, 2009.
- Nagel, R.L., Stone, R.B., and McAdams, D.A., "A process modeling methodology for automation of manual and time dependent processes," *Proc. 2006 ASME IDETC/CIE*, ASME, Philadelphia, PA, 2006.
- Negnevitsky, M., *Artificial Intelligence: A Guide to Intelligent Systems*, Pearson Education, Harlow, England, 2005.
- NNDC, "Interactive chart of nuclides," National Nuclear Data Center, Brookhaven National Lab, <http://www.nndc.bnl.gov/chart>, accessed June 2013.
- Nof, S.Y., *Handbook of Industrial Robotics*, 2nd Ed., John Wiley & Sons, New York, 1999.
- Olvander, J., Lunden, B., and Gavel, H., "A computerized framework for the morphological matrix applied to aircraft conceptual design," *Computer-Aided Design*, Vol. 41, pp. 187-196 (2009).
- Otto, K.N., "Measurement methods for product evaluation," *Research in Engineering Design*, Vol. 7, pp. 86-101 (1995).

- Otto, K.N. and Wood, K.L., "Conceptual and configuration design of products and assemblies," *ASM Handbook, Vol. 20: Materials Selection and Design*, G.E. Dieter (ed.), ASM International, pp. 15-32 (1997).
- Otto, K.N. and Wood, K.L., "Product evolution: a reverse engineering and redesign methodology," *Research in Engineering Design*, Vol. 10, pp. 226-243 (1998).
- Otto, K.N. and Wood, K.L., *Product Design: Techniques in Reverse Engineering and New Product Development*, Prentice-Hall, 2001.
- Pahl, G., Beitz, W., Feldhusen, J., and Grote, K., *Engineering Design: A Systematic Approach*, 3rd ed., Springer-Verlag, London, 2007.
- Pamanes, J. A. and Zeghloul, S., "Optimal placement of robotic manipulators using multiple kinematic criteria," *Proc. 1991 IEEE International Conference on Robotics and Automation*, Sacramento, CA, pp. 933-938, April 1991.
- Papalambros, P.Y. and Wilde, D.J., *Principles of Optimal Design: Modeling and Computation*, Cambridge University Press, Cambridge, UK, 2000.
- Pashkevich, A.P. and Pashkevich, M.A., "Multiobjective optimisation of robot location in a workcell using genetic algorithms," *UKACC International Conference on CONTROL '98*, pp. 757-763, Sept. 1998.
- Pham, D. and Pham, P., "Expert systems in mechanical and manufacturing engineering," *International J. of Adv. Manufact. Tech.*, Vol. 3, No. 3, pp. 3-21 (1988).
- Pholsiri, C., Kapoor, C., and Tesar, D., "Manipulator task-based performance optimization," *Proc. of 2004 ASME 2004 DETC/CIE Conference*, Salt Lake City, UT, September 2004.
- Qian, L. and Gero, J.S., "Function-behavior-structure paths and their role in analogy-based design," *AI for Engineering Design, Analysis and Manufacturing*, Vol. 10, pp. 289-312 (1996).
- Renner, G. and Ekart, A., "Genetic algorithms in computer aided design," *CAD*, Vol. 35, pp. 709-726 (2003).
- Riley, G., "CLIPS: A Tool for Building Expert Systems," <http://clipsrules.sourceforge.net>, accessed April 2013.
- Rinderle, J.R., "Grammatical approaches to engineering design, part II: melding configuration and parametric design using attribute grammars," *Research in Engineering Design*, Vol. 2, pp. 137-146 (1991).
- Rocha, J. and Ramos, C., "Representing and generating operation sequences for manufacturing tasks," *Proc. 1997 IEEE Int. Sym. on Assembly and Task Planning*, IEEE, Marina del Rey, CA, August 1997.

- Rocha, J. and Ramos, C., "Plan representation and generation for manufacturing tasks," *Proc. 1995 IEEE Int. Sym. on Assembly and Task Planning*, IEEE, Pittsburgh, PA, August 1995.
- Rosell, J., Munoz, N., and Gambin, A., "Robot tasks sequence planning using Petri nets," *Proc. of IEEE 5th International Symposium on Assembly and Task Planning*, pp. 24-29, Besançon, France, July 2003.
- Roth, K.H., "Foundation of methodical procedures in design," *Design Studies*, Vol. 2, No. 2, pp. 107-115 (1981).
- Roy, R., Hinduja, S., and Teti, R., "Recent advances in engineering design optimisation: challenges and future trends," *CIRP Annals - Manufact. Tech.*, Vol. 57, pp. 697-715 (2008).
- Rutenbar, R.A., "Simulated annealing algorithms: an overview," *IEEE Circuits and Devices Magazine*, pp. 19-26, January 1989.
- Russell, S. and Norvig, P., *Artificial Intelligence: A Modern Approach*, Prentice Hall, Englewood Cliffs, NJ, 1995.
- Sahbani, A., El-Khoury, S., and Bidaud, P., "An overview of 3D object grasp synthesis algorithms," *Robotics and Autonomous Systems*, Vol. 60, pp. 326-336 (2012).
- Salehi, M. and Tavakkoli-Moghaddam, R., "Application of genetic algorithm to computer-aided process planning in preliminary and detailed planning," *Engineering Applications of Artificial Intelligence*, Vol. 22, pp. 1179-1187 (2009).
- Schmidt, L.C. and Cagan, J., "Optimal configuration design: an integrated approach using grammars," *Journal of Mechanical Design*, Vol. 120, pp. 2-9, March 1998.
- Shea, K., and Starling, A.C., "From discrete structures to mechanical systems: a framework for creating performance-based parametric synthesis tools," In *Proc. of the AAAI 2003 Symposium on Computational Synthesis*, Stanford, CA, pp. 210-217, March 2003.
- Sharp, R. and Decretton, M., "Radiation tolerance of components and materials in nuclear robot applications," *Reliability Engineering and System Safety*, Vol. 53, pp. 291-299 (1996).
- Sheridan, T.B., "Function allocation: algorithm, alchemy or apostasy?" *International Journal of Human-Computer Studies*, Vol. 52, pp. 203-216 (2000).
- Shultis, J.K. and Faw, R.E., *Radiation Shielding*, American Nuclear Society, La Grange Park, IL, 2000.
- Sim, S.K. and Duffy A.H.B., "Towards an ontology of generic engineering design activities," *Research in Engineering Design*, Vol. 14, pp. 200-223 (2003).

- Singh, S.P. and Sharma, R.R.K., "A review of different approaches to the facility layout problems," *International Journal of Advanced Manufacturing Technology*, Vol. 30, pp. 425-433 (2006).
- Souilah, A., "Simulated annealing for manufacturing systems layout design," *European Journal of Operational Research*, Vol. 83, pp. 592-614 (1995).
- Sridharan, P. and Campbell, M.I., "A study on the grammatical construction of function structures," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Vol. 19, No. 3, pp. 139-160 (2005).
- Strawbridge, Z., McAdams, D, and Stone, R., "A computational approach to conceptual design," *Proc. 2002 ASME DETC*, ASME, Montreal, Canada, 2002.
- Stiny, G. and Gips, J., "Shape grammars and the generative specification of painting and sculpture," in *The Best Computer Papers of 1971*, Ed. O.R. Petrocelli, Auerbach, Philadelphia, pp. 125-135 (1972).
- Stiny, G., "Introduction to shape and shape grammars," *Environment and Planning B*, Vol. 7, pp. 343-351 (1980).
- Stoll, H.W., "Introduction to manufacturing and design," *ASM Handbook, Vol. 20: Materials Selection and Design*, G.E. Dieter (ed.), ASM International, pp. 669-675 (1997).
- Stone, R.B., Wood, K.L., and Crawford, R.H., "Using quantitative functional models to develop product architectures," *Design Studies*, Vol. 21, pp. 239-260 (2000).
- Suh, N.P., Cochran, D.S., and Lima, P.C., "Manufacturing system design," *Annals of the CIRP*, Vol. 47, No. 2, pp. 627-639 (1998).
- Suh, N.P., *The Principles of Design*, Oxford University Press, 1990.
- Szykman, S. and Cagan, J., "Constrained three-dimensional component layout using simulated annealing," *Journal of Mechanical Design*, Vol. 119, pp. 28-35 (1997).
- Tay, M.L. and Ngoi, B.K.A., "Optimizing robot workcell layout," *International Journal of Advanced Manufacturing Technology*, Vol. 12, pp. 377-385 (1996).
- Theodoracatos, V.E. and Ahmed, M.F., "ECDEX: a knowledge-based approach to conceptual design of engineering systems," *Robotics & Computer-Integrated Manufacturing*, Vol. 11, No. 3, pp. 137-166 (1994).
- Thurston, D.L., "A formal method for subjective design evaluation with multiple attributes," *Research in Engineering Design*, Vol. 3, pp. 105-122 (1991).
- Tisius, M., Pryor, M., Kapoor, C., and Tesar, D., "An empirical approach to performance criteria for manipulation," *Journal of Mechanisms and Robots*, Vol. 1, No. 3, (2009).

- Torczon, V. and Trosset, M.W., "From evolutionary operation to parallel direct search: pattern search algorithms for numerical optimization," *Comput. Sci. Statist.*, Vol. 29, pp. 396-401 (1997a).
- Torczon, V., "On the convergence of pattern search algorithms," *SIAM J. Optimization*, Vol. 7, No. 1, pp. 1-25 (1997b).
- Triantaphyllou, E., *Multi-Criteria Decision Making Methods: A Comparative Study*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000.
- Ullman, D.G., *The Mechanical Design Process*, McGraw-Hill, New York, 1992.
- Umeda, Y. and Tomiyama, T., "FBS modeling: modeling scheme of function for conceptual design," *Workshop on Qualitative Reasoning about Physical Systems*, pp. 271-278 (1995).
- Venkatachalam, A.R., Mellichamp, J.M., and Miller, D.M., "A knowledge-based approach to design for manufacturability," *Journal of Intelligent Manufacturing*, Vol. 4, pp. 355-366 (1993).
- Wang, Y. and Duarte, J.P., "Automatic generation and fabrication of designs," *Automation in Construction*, Vol. 11, pp. 291-302 (2002).
- Ward, A.C. and Seering, W.P., "The performance of a mechanical design compiler," *ASME Design Engineering*, Vol. 17, pp. 89-97 (1989).
- Williams, J.M., "Improved manipulator configurations for grasping and task completion based on manipulability," *Master's Thesis*, The Department of Mechanical Engineering, The University of Texas at Austin, December 2010.
- Woldemichael, D.E. and Hashim, F.M., "A framework for function-based conceptual design support system," *J. of Eng., Design and Tech.*, Vol. 9, No. 3, pp. 250-272 (2011).
- Yang, J. and Abdel-Malek, K., "Human reach envelope and zone differentiation for ergonomic design," *Human Factors and Ergonomics in Manufacturing*, Vol. 19, No. 1, pp. 15-34 (2009).
- Yi, M., Fadel, G.M., and Gantovnik, V.B., "Vehicle design with a packing genetic algorithm," *International Journal of Heavy Vehicle Systems*, Vol. 15, No. 2/3/4, pp. 433-448 (2008).
- Yin, S. and Cagan, J., "Exploring the effectiveness of various patterns in an extended pattern search layout algorithm," *Journal of Mechanical Design*, Vol. 126, pp. 22-28 (2004).
- Yin, S. and Cagan, J., "An extended pattern search algorithm for three-dimensional component layout," *Journal of Mechanical Design*, Vol. 122, pp. 102-108 (2000).

- Yoshikawa, T., "Manipulability of robotic mechanisms," *Robotics Research: The Second International Symposium*, Eds. H. Hanafusa and H. Inoue, Cambridge: MIT Press, pp. 439-446, August 1985.
- Youk, G.U., Lee, N.H., Kim, B.S., Lee, Y.B., and Kim, S., "Technology development for the radiation hardening of robots," *Proc. 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol. 3, pp. 1715-1720 (1999).
- Zacharias, F., Borst, C., and Hirzinger, G., "Capturing robot workspace structure: representing robot capabilities," *Proc. of 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Diego, CA, October 2007.
- Zandin, K.B., *MOST Work Measurement Systems*, 3rd Ed., Marcel Dekker, New York, 2002.
- Zha, X.F., Du, H.J., and Qiu, J.H., "Knowledge-based approach and system for assembly oriented design, Part I: the approach," *Engineering Applications of Artificial Intelligence*, Vol. 14, pp. 61-75 (2001).
- Zhang, W., Tor, S., and Britton, G., "Automated functional design of engineering systems," *J. of Intelligent Manufact.*, Vol. 13, pp. 119-133 (2002).
- Zwicky, F., "The morphological method of analysis and construction," *Courant Anniversary Volume*, Intersciences Publishers, New York, pp. 461-470, 1948.

Vita

Joshua Murry Williams was born in Phoenix, AZ, in 1985. He grew up in Humboldt, AZ, and attended Bradshaw Mountain High School, where he graduated as Co-Valedictorian in 2004. Joshua attended Occidental College in Los Angeles, graduating with honors and Magna Cum Laude in 2008 with a B.A. in Physics. He entered the Mechanical Engineering program at The University of Texas at Austin to study nuclear engineering and robotics and received his M.S.E. in December 2010. Joshua currently resides in northern New Mexico.

Email: jmwilliams@utexas.edu

This dissertation was typed by the author.